



HAL
open science

A generic approach towards the collaborative construction of digital scholarly editions

Vincent Barrellon

► **To cite this version:**

Vincent Barrellon. A generic approach towards the collaborative construction of digital scholarly editions. Document and Text Processing. Université de Lyon, 2017. English. NNT : 2017LYSEI113 . tel-02090792

HAL Id: tel-02090792

<https://theses.hal.science/tel-02090792v1>

Submitted on 5 Apr 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



INSA

N°d'ordre NNT : 2017LYSEI113

THESE de DOCTORAT DE L'UNIVERSITE DE LYON
opérée au sein de
INSA LYON

Ecole Doctorale N° EDA 512
Ecole doctorale d'Informatique et de Mathématiques de Lyon

Spécialité du doctorat : Informatique

Soutenue publiquement le 27/11/2017, par :
Vincent Barrellon

A Generic Approach towards the Collaborative Construction of Digital Scholarly Editions

Devant le jury composé de :

Muriasco, Elisabeth	Pr	Université de Toulon	Rapporteuse
Munson, Ethan	Pr	University of Wisconsin-Milwaukee	Rapporteur
Pierazzo, Elena	Pr	Université Grenoble Alpes	Examinatrice
Vion-Dury, Jean-Yves	Dr	Naver Labs Europe	Examineur
Calabretto, Sylvie	Pr	INSA Lyon	Directrice de thèse
Portier, Pierre-Edouard	MdC	INSA Lyon	Co-directeur de thèse
Ferret, Olivier	Pr	Université Lyon 2	Co-encadrant

Département FEDORA – INSA Lyon - Ecoles Doctorales – Quinquennal 2016-2020

SIGLE	ECOLE DOCTORALE	NOM ET COORDONNEES DU RESPONSABLE
CHIMIE	CHIMIE DE LYON http://www.edchimie-lyon.fr Sec : Renée EL MELHEM Bat Blaise Pascal 3 ^e etage secretariat@edchimie-lyon.fr Insa : R. GOURDON	M. Stéphane DANIELE Institut de Recherches sur la Catalyse et l'Environnement de Lyon IRCELYON-UMR 5256 Équipe CDFA 2 avenue Albert Einstein 69626 Villeurbanne cedex directeur@edchimie-lyon.fr
E.E.A.	ELECTRONIQUE, ELECTROTECHNIQUE, AUTOMATIQUE http://edeea.ec-lyon.fr Sec : M.C. HAVGOUDOUKIAN Ecole-Doctorale.eea@ec-lyon.fr	M. Gérard SCORLETTI Ecole Centrale de Lyon 36 avenue Guy de Collongue 69134 ECULLY Tél : 04.72.18 60.97 Fax : 04 78 43 37 17 Gerard.scorletti@ec-lyon.fr
E2M2	EVOLUTION, ECOSYSTEME, MICROBIOLOGIE, MODELISATION http://e2m2.universite-lyon.fr Sec : Sylvie ROBERJOT Bât Atrium - UCB Lyon 1 04.72.44.83.62 Insa : H. CHARLES secretariat.e2m2@univ-lyon1.fr	M. Fabrice CORDEY CNRS UMR 5276 Lab. de géologie de Lyon Université Claude Bernard Lyon 1 Bât Géode 2 rue Raphaël Dubois 69622 VILLEURBANNE Cédex Tél : 06.07.53.89.13 cordey@univ-lyon1.fr
EDISS	INTERDISCIPLINAIRE SCIENCES-SANTE http://www.ediss-lyon.fr Sec : Sylvie ROBERJOT Bât Atrium - UCB Lyon 1 04.72.44.83.62 Insa : M. LAGARDE secretariat.ediss@univ-lyon1.fr	Mme Emmanuelle CANET-SOULAS INSERM U1060, CarMeN lab, Univ. Lyon 1 Bâtiment IMBL 11 avenue Jean Capelle INSA de Lyon 696621 Villeurbanne Tél : 04.72.68.49.09 Fax :04 72 68 49 16 Emmanuelle.canet@univ-lyon1.fr
INFOMATHS	INFORMATIQUE ET MATHEMATIQUES http://infomaths.univ-lyon1.fr Sec : Renée EL MELHEM Bat Blaise Pascal, 3 ^e étage Tél : 04.72. 43. 80. 46 Fax : 04.72.43.16.87 infomaths@univ-lyon1.fr	M. Luca ZAMBONI Bâtiment Braconnier 43 Boulevard du 11 novembre 1918 69622 VILLEURBANNE Cedex Tél :04 26 23 45 52 zamboni@maths.univ-lyon1.fr
Matériaux	MATERIAUX DE LYON http://ed34.universite-lyon.fr Sec : Marion COMBE Tél:04-72-43-71-70 –Fax : 87.12 Bat. Direction ed.materiaux@insa-lyon.fr	M. Jean-Yves BUFFIERE INSA de Lyon MATEIS Bâtiment Saint Exupéry 7 avenue Jean Capelle 69621 VILLEURBANNE Cedex Tél : 04.72.43 71.70 Fax 04 72 43 85 28 Ed.materiaux@insa-lyon.fr
MEGA	MECANIQUE,ENERGETIQUE,GENIE CIVIL,ACOUSTIQUE http://mega.universite-lyon.fr Sec : Marion COMBE Tél:04-72-43-71-70 –Fax : 87.12 Bat. Direction mega@insa-lyon.fr	M. Philippe BOISSE INSA de Lyon Laboratoire LAMCOS Bâtiment Jacquard 25 bis avenue Jean Capelle 69621 VILLEURBANNE Cedex Tél : 04.72 .43.71.70 Fax : 04 72 43 72 37 Philippe.boisse@insa-lyon.fr
ScSo	ScSo* http://recherche.univ-lyon2.fr/scso/ Sec : Viviane POLSINELLI Brigitte DUBOIS Insa : J.Y. TOUSSAINT Tél : 04 78 69 72 76 viviane.polsinelli@univ-lyon2.fr	M. Christian MONTES Université Lyon 2 86 rue Pasteur 69365 LYON Cedex 07 Christian.montes@univ-lyon2.fr

*ScSo : Histoire, Géographie, Aménagement, Urbanisme, Archéologie, Science politique, Sociologie, Anthropologie

A Generic Approach towards
the Collaborative Construction
of Digital Scholarly Editions

A Manel et Sacha.

Remerciements

Avant de plonger dans le dur, je souhaite prendre quelques lignes, dans ma langue natale, pour adresser mes profonds remerciements à celles et ceux qui ont contribué, de mille façons, à ce travail et à ces quatre belles années.

Je remercie en premier lieu Sylvie, pour sa bienveillance, son invincible gentillesse, son pilotage, pour son soutien et son agilité à trouver des financements de dernière minute, sans lesquels j'aurais mangé beaucoup de riz blanc ces douze derniers mois.

Je remercie très vivement Olivier, tout d'abord pour les immenses efforts déployés tout au long de cette thèse afin d'en suivre le moindre détail, jusque dans ses confins parfois bien éloignés de la littérature, mais aussi pour avoir été un soutien moral important dans des moments clés, et pour m'avoir permis de me perfectionner dans l'art de la réunion de travail, à la Croix-Rousse.

Je remercie aussi, tout particulièrement, Pierre-Edouard, avec qui la littérature, le cinéma, la philosophie, les cookies et le bon café sont souvent le présentiment ou le corollaire d'une découverte ; je le remercie pour les innombrables discussions fouillées, exigeantes, enthousiasmantes, extraordinaires qui ont jalonné ces quatre années de thèse.

Je remercie aussi les membres de mon jury, pour le temps consacré à la lecture de ce travail, au voyage, et à la discussion finale, le jour j^1 .

Je remercie les membres de mon laboratoire, le LIRIS, et plus particulièrement des équipes DRIM et BD, avec qui j'ai eu davantage l'occasion de faire connaissance et de nouer, parfois, de belles relations d'amitiés.

¹Je présente mes excuses pour introduire des noms de variables jusque dans les remerciements de ce mémoire. Une cure de désintoxication semble nécessaire.

Abstract

Digital Scholarly Editions are critically annotated patrimonial literary resources, in a digital form. Such editions roughly take the shape of a transcription of the original resources, augmented with critical information, that is, of structured data. In a collaborative setting, the structure of the data is explicitly defined in a schema, an interpretable document that governs the way editors annotate the original resources and guarantees they follow a common editorial policy.

Digital editorial projects classically face two technical problems. The first has to do with the expressiveness of the annotation languages, that prevents from expressing some kinds of information. The second relies in the fact that, historically, schemas of long-running digital edition projects have to evolve during the lifespan of the project. However, amending a schema implies to update the structured data that has been produced, which is done either by hand, by means of *ad-hoc* scripts, or abandoned by lack of technical skills or human resources.

In this work, we define the theoretical ground for an annotation system dedicated to scholarly edition. We define eAG, a stand-off annotation model based on a cyclic graph model, enabling the widest range of annotation. We define a novel schema language, SeAG, that permits to validate eAG documents on-the-fly, while they are being manufactured. We also define an inline markup syntax for eAG, reminiscent of the classic annotation languages like XML, but retaining the expressivity of eAG. Eventually, we propose a bidirectional algebra for eAG documents so that, when a SeAG S is amended, giving S' , an eAG I validated by S is semi-automatically translated into an eAG I' validated by S' , and so that any modification applied to I (resp. I') is semi-automatically propagated to I' (resp. I) – hence working as an assistance tool for the evolution of SeAG schemas and eAG annotations.

Résumé

Dans la continuité des éditions critiques traditionnelles, les années 2000 ont vu l'émergence de nouveaux objets éditoriaux : les éditions critiques numériques. De telles éditions se distinguent des numérisations de fonds patrimoniaux en offrant une transcription enrichie de données secondaires (notes et commentaires, indication des sources, données bio/bibliographiques, etc.). De tels objets prennent la forme de documents structurés, e.g. de documents XML. En pratique, les éditions critiques s'appuient sur une structure de données définie explicitement sous la forme d'un schéma. Reflet de la politique éditoriale, le schéma indique le vocabulaire et la grammaire d'annotation mis en œuvre dans l'édition.

Si XML est le standard *de facto* pour l'annotation textuelle, il est établi qu'une structure hiérarchique représente mal un texte littéraire, et a fortiori un texte annoté. D'autres modèles d'annotation, reposant sur des formalismes de graphes plus généraux, ont été proposés : les modèles de documents multist structurés. Cependant, les mécanismes de validation pour documents multist structurés échouent à embrasser la catégorie de graphes la plus prometteuse pour la réalisation d'annotations, à savoir des graphes cycliques, du moins avec une complexité algorithmique raisonnable.

Sur un autre plan, il apparaît que les projets d'édition mettent en œuvre un mode de travail linéaire, comme suit : un schéma est initialement proposé ; il est dans un second temps instancié, puis les documents résultant de cette instanciation sont publiés. Le problème pratique est que les éditeurs, dans la phase de conception du schéma, anticipent rarement sur l'ensemble des situations qui seront rencontrées dans les textes annotés. En d'autres termes, les schémas sont appelés à évoluer pendant la construction de l'édition. Or une mise à jour du schéma doit s'accompagner d'une mise à jour des données structurées...

Nos travaux s'articulent autour de trois axes : la représentation d'annotations critiques concurrentes ou multist structurées ; la validation de telles données ; l'assistance à la mise à jour de ces données en cas d'amendement du schéma correspondant.

1. Nous avons défini un modèle de balisage déporté (stand-off markup), nommé eAG, dans lequel l'annotation est représentée sous la forme d'un graphe cyclique. Nous avons défini et formalisé la notion de chronologie composite, permettant de faire référence aux contenus de documents comportant, typiquement, du texte et de l'image, et à rendre ce type de documents compatible avec le balisage déporté. Nous avons proposé une syntaxe de balisage pour l'annotation textuelle, nommée LeAG.

2. La validation de données dont la structure, comme dans les eAG, est un graphe cyclique, est coûteuse si elle repose sur une grammaire (RelaxNG, Creole, Schex, etc.). A ce titre, nous proposons la notion de simulation en tant que mécanisme de validation. Un schéma SeAG est un graphe qui valide un eAG si il simule ce dernier. Nous avons défini un mode de représentation couplé pour les schémas et les graphes d'annotation tel que, étant donné la représentation d'un schéma, seuls des graphes d'annotation simulés par le schéma puissent être représentés, rendant possible une validation « par construction » des eAG.
3. Le troisième axe de recherche s'apparente à la problématique des transformations bidirectionnelles ou du Data Exchange dans laquelle, en fonction de correspondances entre deux schémas, les données instanciant le premier schéma sont traduites dans une forme compatible avec le second. Notre approche se base sur un petit nombre de primitives, opérations élémentaires qui s'appliquent aux schémas : suppression, insertion, substitution d'une sous-partie. Un schéma est amendé par application successive de ces primitives. Chacune de ces primitives, en outre, peut être interprétée au niveau des instances et donner une nouvelle forme d'instance, compatible avec le schéma modifié. Enfin, les transformations entre instances sont bidirectionnelles, assurant une synchronisation entre les instances de deux schémas.

Acknowledgement

This PhD was supported by the ARC5 program of the Rhône-Alpes region, France.



Contents

I	Introduction	1
1	Digital Scholarly Edition	3
1.1	Edition and Text Theory in the Digital World	4
1.2	Constructing Digital Scholarly Editions: A Generic Approach	10
1.2.1	A Generic Approach of DSE	12
1.2.1.1	A Starting Model for DSE Manufacture	12
1.2.1.2	A Composite Approach of Genericness	13
1.2.1.3	Conclusion of this Paragraph	15
1.2.2	Data Structuring Models	16
1.2.2.1	A Panorama of Data Structuring Paradigms	16
1.2.2.2	Making the Structure Evolve	19
1.2.2.3	Intermediate Summary	26
1.2.3	Collaborative Data Structuring in DSE Projects	26
1.2.3.1	The Paradoxical Needs of Collaborative DSE Teams	26
1.2.3.2	Conclusion: Paradoxical Needs	30
2	Proposition of a Collaborative Construction Process	31
2.1	Current Approaches of the Problem	32
2.1.0.3	Altruistic vs. Egoistic Data Structures	32
2.1.0.4	The Theory of Common Ground	34
2.2	Operating the Common Ground Theory	38
2.2.1	The Common Ground Theory: Applications	39
2.2.2	CG-inspired collaborative data structuring in a DSE setting	42
2.2.2.1	A Double Interpretation of the Common Ground	42
2.2.2.2	CG-inspired Data Structuring: Illustration	43
2.3	Conclusion, Stakes and Challenges	47
II	The eAG/SeAG data model	53
3	Introduction	55
3.1	Preliminary: Notation	55
3.2	Outline of this Part and Main Contributions	56

4	Related Work	59
4.1	Multistructured Data and Validation	59
4.1.1	Multistructured Data Models	60
4.1.2	M-S Validation: Algorithmic Complexity	61
4.2	The Annotation Graphs Model	63
4.2.1	Annotation Graphs	64
4.2.2	Criticisms	68
5	Extended Annotation Graphs and Schema models	71
5.1	The Extended Annotation Graphs Model	71
5.1.1	An Example of eAG Annotation: Anaphoric Chains	71
5.1.2	The eAG Model, Formally	75
5.1.2.1	eAG Graph Model	75
5.1.2.2	Authorized Labels in eAG	76
5.1.2.3	Chronologies in eAG	76
5.1.2.4	Elements, Hierarchies and Links in eAG	79
5.1.2.5	Elements in an eAG: Precisions	88
5.1.3	Conclusion	90
5.2	Schema Model	91
5.2.1	Finite-State Machine Analogy	92
5.2.1.1	The Notion of “Language of Annotation”	92
5.2.1.2	Regular Expression-based Language Representation	93
5.2.1.3	Language of an eAG: Interpretation	94
5.2.2	The SeAG Model, Formally	97
5.2.2.1	SeAG Graph Model and Instantiation Function	97
5.2.2.2	Schema-Instance Relation: Node-typed Simulation	99
5.2.2.3	SeAG Expressive Power	102
5.2.2.4	SeAG Expressive Power: Anaphoric Chains Validation	106
5.2.2.5	Simulation-based Validation: Caveats	108
5.2.2.6	Grammar vs. Simulation-based Validation	108
5.2.3	Precisions on SeAG	110
5.2.3.1	Multiple forms for the same schema?	110
5.2.3.2	Redundancy and ambiguity	113
5.3	SeAG Validation: A Posteriori and On-the-fly Validations	119
5.3.0.3	A Posteriori Validation	119
5.3.0.4	On-the-fly Validation	120
5.4	Conclusion	124
III	LeAG: an Inline Markup Syntax for eAG	125
6	Inline Multilayer Annotation	127
6.1	Introduction	127
6.2	The LeAG Syntax	129
6.2.1	Mono-hierarchy of Attributeless Elements	129
6.2.2	Grafts: Multilayer Annotation	130

6.2.2.1	Colouring the annotation layers: general strategy . . .	131
6.2.2.2	Positioning Colour Tags	134
6.2.3	Standard Inserts: Attributes, Structured Comment	137
6.2.4	Links and Quoting Elements	139
6.3	Summary and Notation	142
7	An Efficient Parser for Linear Extended Annotation Graphs	145
7.1	General Parsing Strategy	146
7.2	Parsing Strategy: Elements of Design	147
7.2.1	Restrictions on ESE Defining Tags	148
7.2.2	Restrictions on ϵ Edges	154
7.2.3	Restrictions on Insert Tags	157
7.2.3.1	Comment Insert Tags: Parsing Strategy.	167
7.2.3.2	Quote Insert Tags: Parsing Strategy.	169
7.2.3.3	Attribute Insert Tags: Parsing Strategy.	170
7.2.3.4	Void Inserts: Parsing Strategy	170
7.2.3.5	Link Inserts: Parsing Strategy	170
7.2.3.6	Summary	171
7.2.4	Connecting ToT Graphs: Ongoing Hierarchies of Elements . . .	172
7.2.4.1	Pending Nodes	173
7.2.4.2	Non-Pending Nodes	173
7.2.5	Connecting ToT Graphs: Colour Tags Handling	175
7.2.5.1	Case 1	177
7.2.5.2	Case 2	180
7.3	Parsing Algorithm	191
7.3.1	Data Structures	191
7.3.2	Parsing Algorithm	196
7.3.2.1	Main Algorithm	196
7.3.2.2	Associate Nodes to Tags	201
7.3.2.3	Connect the Connectible Nodes of a Hierarchical Level	201
7.3.2.4	Insert with LeAG/ID ₂ Field Parsing	203
7.3.2.5	Target Node Association	205
7.3.3	Parsing Algorithm: Time Complexity	205
7.4	Conclusion	209
IV	Bidirectionalizing eAG/SeAG	211
8	Introduction	213
8.1	The Problem to Solve	213
8.2	Schema Evolution, Bidirectional Transformations	214
8.2.1	Schema Evolution in Database Studies	214
8.2.2	Bidirectional Transformations	216
8.2.3	Bidirectionalizing eAG/SeAG	218

9	SeAG Transformations	219
9.1	Matrix-based Representation of eAG/SeAG: Calculability	219
9.2	Composing Modifications: General Strategy	223
9.3	<i>Mod</i> Operator	224
9.3.1	Schematic Cells	224
9.3.2	<i>Mod</i> Operator: Intuitive Presentation	226
9.3.3	<i>Mod</i> : Formal Definition in the General Case	228
9.3.3.1	Non Independent Schematic Cells	228
9.3.3.2	Connectivity Factor	230
9.3.3.3	<i>Mod</i> for Partially Independent Schematic Cells	235
9.3.3.4	<i>Mod</i> for Ambiguous Schematic Cells	238
9.3.3.5	Notation	243
9.4	<i>Split</i> Operator	243
9.5	<i>Unite</i> Operator	246
9.5.1	Situations where the <i>Unite</i> Operator Applies	246
9.5.2	<i>Unite</i> Operator Definition	248
9.6	Operators Composability	250
10	eAG Bidirectional Transformations	253
10.1	Composing eAG Modifications	253
10.2	Definition and Temporal Model of Instance Update	254
10.3	Derivation from a <i>Mod</i> Modification	257
10.3.1	eAG <i>Mod</i> Behaviour	257
10.3.2	Bidirectional eAG <i>Mod</i>	259
10.4	Derivation from a <i>Split</i> Modification	274
10.5	Derivation from a <i>Unite</i> Modification	274
10.5.1	Forward Derivation	276
10.5.2	Backwards Derivation	281
10.6	Reference Values Propagation	292
10.7	Composing Modifications: Two Quick Examples	294
10.7.1	First Example.	294
10.7.2	Second Example: Making Some Pattern Cyclic	295
10.8	Conclusion	295
V	General conclusion	299
10.9	Contributions	301
10.10A	Word on the Adopted Methodology	303
10.11	Future Work and Perspectives	304
	Bibliography	307
	Appendix	323
11	eAG Example: Multiple Chronology-based Annotation.	323

<i>CONTENTS</i>	xvii
12 Tree-Automata vs. Simulation-based Validation	327
13 eAG <i>Mod</i> Operation: General Case	341
13.1 Pavage	342
13.2 Permutations entre matrices d'une même instance	344
13.3 Alignement	346

List of Figures

1.1	Elementary model for DSE construction.	14
1.2	The working surface of CritSpace. The information that two documents share close relationship is indicated by the spatial closeness of the graphical representation of those documents.	18
1.3	Representation of the data structure Alpha, instantiated on the digital corpus A.	21
1.4	Representation of the data structure <i>Bêta</i> , instantiated on the digital corpus <i>B</i>	23
1.5	Synthetic view of the conclusions of Paragraph 1.2.3.1 highlighting the contradictions between those conclusions.	30
2.1	The theory of <i>Common Ground</i> relies upon two notions: grounding, process by means of which a certain impression of mutual understanding is acquired by the coactors; grounding criterion, referring to the evidence or clues of mutual understanding that punctuate the interaction.	36
4.1	The classic multistructured data models, classified by the family of graphs of elements they enable to express.	62
4.2	Possible configurations of an annotation layer in the Annotation graph model (extracted from [21], p. 40.)	69
5.1	An adapted passage from <i>The Village of Ben Suc</i> by J. Schell, with some highlighted anaphoric chains and constituting singular expressions.	72
5.2	Document showing overlap, a figure enclosed in text and internal references.	76
5.3	An eAG representing Figure 1 and a way to browse through it (arrows). Grey edges are for reading assistance (they span over the paths defining a structured element). Speech balloons show reference values (shades differentiate between chronometers), from a chronology extending $\langle T, \leq \rangle$ (cf. <i>Illustration, part 1</i>) in order to detail the content of Page one and a Ref (“[a]” in the text) between characters 150 and 153.	85
5.4	SeAG schema validating the eAG given in Figure 5.3.	97

5.5	Two fundamental properties of simulation-based validation. a. Two SeAG S_A and S_B patterns in parallel may validate either any instance of one of the patterns, or any superposition of instances of one of the patterns. b. A SeAG pattern S made cyclic will validate any concatenation of any of its instances, provided the concatenation is well-formed, or any superposition of such concatenations.	104
5.6	RelaxNG tree automaton-based XML validation mechanism.	109
5.7	For all $n \in N$ as defined in Figure 5.6, each n Box representation (middle) and $R_n \subset R$ (right).	110
5.8	A non-redundant but ambiguous graph.	113
6.1	A basic anaphoric chain annotation for the extract of <i>The Village of Ben Suc</i> , and a corresponding schema.	129
6.2	A LeAG and a matching eAG, where an element (B) has two fathers, one in the uncoloured hierarchy, and the other in a graft. The colours of B, namely #G and # (uncoloured), are repeated in the context of its son element α	133
6.3	Three-layered LeAG.	133
7.1	Two corresponding SeAG/LeAG/eAG triples, illustrating (a) Simulation-based multilayering and (b) Schema-based multilayering. The coloured areas highlight the correspondence between the ToTs and the elements they define.	176
7.2	Parsing strategy for the LeAG $\{A \text{ in } X\} [\#R \text{ over } X] [C \text{ in } \#R] [B \text{ in } X]$, with schemas so that the starting node of the graft and the root of its first element are one same node.	178
7.3	Parsing strategy for the LeAG $[A \text{ in } X] [\#R \text{ over } X] [C \text{ in } \#R] [B \text{ in } X]$, with schemas so that the starting node of the graft and the root of its first element are not the same node.	179
7.4	(a) Catalogue of the possible situations in which more than one node may belong to the same hierarchical level "in X". (b) The typology above shows that just knowing the complete value of the hierarchical level of two nodes, in the shape "in X, A, B", is enough to assess which of the two nodes will precede the other in the eAG graph. Precedence is represented by the dotted arrows. The precedence relation is transitive.	183
7.5	The ordered list of possible <i>items</i> in a given hierarchical level (vertical, green) and for each, the ordered list of possible <i>targets</i> (horizontal, pink).	204
8.1	The Data Exchange problem. Figure extracted from [108].	216
10.1	Summary of the editorial data system we intended to design (left), and the one we have defined (right).	302

List of Tables

1.1	Typology of the different kinds of data structure amendments, and the resulting data conflicts.	25
5.1	Allowed suffixes per label class and the resulting class. “-” stands for “undefined”.	77
5.2	The different kinds of eAG elements, according to the label class their name belongs to.	80
6.1	Summary of the syntax for LeAG tags.	143
7.1	Conditions for two nodes to be compatible for being part of the same pair in the data structure E.	201
7.2	List of the primary and secondary cells of the structure HL, relative to a given tag.	202
9.1	The term-to-term addition operated for summing two incidence matrices. This operator ensures to keep the result of any sum of values from the set $\{-1, 0, 1\}$, which is the set of values of the elements of an incidence matrix, in the same set.	223
12.1	Bilan des sous-graphes de S définis par récurrence horizontale, par ordre décroissant sur l’inclusion ($S_1 \supseteq S_1^+$ etc.), et des hiérarchies correspondantes.	338

Part I

Introduction

Chapter 1

Digital Scholarly Edition

Scholarly Editing, or the art of making “available for scholarly use works¹ not ordinarily available or available only in corrupt or inadequate forms” [155], has been one of the paramount activities of scholars for centuries. Resulting from long-term, close-study of the edited corpus, scholarly editions provide the reader with an appropriate representation of the primary corpus, or, in case this corpus is lost, with a reasoned reconstruction of it; it may also, depending on the scholar’s editorial strategy, give detailed insights of the genesis of the edited works and/or their transmission and variation. Textual introductions and notes may also be added to the edition, in order to supply, either at the global or at the local scale of the corpus, contextual or explicative information. Thus, scholarly editions are among the most complex editorial objects that the codex tradition have given birth to.

Established and evaluated according to the academic standards [106], scholarly editions are fed by original or up-to-date studies on the edited corpus – as such, editing “produces knowledge” [142] and editions represent important, milestone *research products*. Conversely, because they are expected to represent the state-of-the-art knowledge about the edited work, at the moment of their publication, scholarly editions are key resources for further research, that is, valuable *research tool* themselves. In other words, scholarly editions plays a pivotal role in the Humanities, aggregating prior knowledge and opening to new discoveries at the same time.

It is thus no wonder that Digital Scholarly Editions (DSE) shall be one of the most active fields in the frame of the booming Digital Humanities. Quantitative evidence of the intense editorial activity among the digital scholars can be found in the Patrick Sahle Catalog, that counts as many as 412 DSE projects undergone over the period starting in 1994 until this very day^{2,3}. Digital Humanities can be defined, in the view

¹Or texts, or documents.

²Available online at <http://www.digitale-edition.de/>. Accessed on August 1st, 2017.

³One may notice, while browsing through the catalogue, the drastic under-representation of DSE projects conducted outside the ‘Western world’. Actually, the only counter-example from the Sahle Catalog is the *Bichitra: Online Tagore Variorum* project, conducted at the Jadavpur University, Kolkata, India. This under-representation, yet, seems to represent quite accurately the global DSE landscape. Indeed, as evidenced by [74], DSE is mainly a European and Anglo-Saxon phenomenon, and very few such projects are actually undergone for non-European language-based corpora. Arabic

of some, by their vocation to become a digital research infrastructure for humanists, in an analogous way to the infrastructure that libraries, universities, and so on, constitute in the physical world [61]. Analogically indeed, DSE aim at being both a research product and tool; analogically, DSE are established and evaluated according to the academic standards – as an illustration, the peer-review RIDE Journal precisely aims to “direct attention to digital editions and to provide a forum in which expert peers criticise and discuss the efforts of digital editors in order to improve current practices and advance future developments”⁴. And certainly, shall DSE have been no more than the transposition of the traditional scholarly edition in that digital infrastructure, it would indeed have had to play a central role in that digital setting.

Still, beyond these analogical points between paper-based scholarly edition and DSE, the prospects of DSE go far beyond working as a sort of digital incubator for scholarly editions. Actually, it seems that going digital profoundly is renewing the scholars’s activity. Some of the traditional, theoretical problems the editors have faced for centuries are renewed by the shift from the codex shape to the digital world; new resources (multimedia resources in particular) can be edited, or can be integrated into the critical apparatus of a DSE, opening new perspectives in terms of editing capacities; alternative business models develop along the traditional ones, based on the publishing industry; eventually, new working organisation models (collaborative work) are at hand, enabling to envision the edition of vast corpora, or multidisciplinary, or international edition projects, in a way that was much harder to follow without digital support. Digital edition also raised new, tricky problems; in particular, long-term preservation is a key challenge, as it is for Digital Humanities as a whole.

In this introduction, before dwelling into the technical landscape underlying DSE, we would like to focus on how DSE are theorized, modelled, envisioned by the editors themselves, and on the practical, general implications of those visions. This may help understand what DSE are or could be.

In a second part, we will investigate the notion of DSE construction, in the light of the above. We will then ponder what a generic approach towards the construction of DSE might mean.

1.1 Edition and Text Theory in the Digital World

Scholarly Edition⁵ is traditionally divided in two main fields: **critical** versus **documentary** edition. Indeed, the two kinds do complement one another, from a definitional point of view: while critical edition refers to “a text that derives from more than one source text” [155], a documentary edition can be defined as “an edition of a text

and Asian languages are virtually short of examples to this date. Interestingly, the only three scholarly digital projects dedicated to Japanese corpora that we are aware of are based outside Japan, namely: the *Japan Text Initiative*, from the University of Virginia, which is a collection of copy-texts; the *Japanese Historical Text Initiative*, which is a database of copy-texts and translations, from the University of Berkeley ; the *Oxford Corpus of Old Japanese*, exhibiting more ambitious scholarly goals [75], edited at the Oxford University.

⁴Accessible at <http://ride.i-d-e.de/>. Accessed on August 4th 2017.

⁵Many definitions cited here come from the very useful Lexicon of Scholarly Editing, founded by Dirk Van Hulle, available at uahost.uantwerpen.be/lse/. Accessed on August 5th, 2017.

based on a single document” [135]. They also correspond to different rationale, and result in very diverse objects.

Documentary edition “attempts to reproduce a certain degree of the peculiarities of the document itself, even if this may cause disruption to the normal flow of the text presented by the document. It can assume different formats, by presenting the textual content of the document as semi-diplomatic, diplomatic, ultra-diplomatic, or even facsimile editions, which differentiate themselves by the level of editorial intervention, ranging from the largest to the smallest concession to the reading habits of the public of choice” [135].

Critical edition, on the opposite, as suggested by the above, minimal definition, implies that the editor methodologically *constructs* the text of the edition, based on all the available documents that constitute her corpus (the selected set of historical ‘witnesses’, or versions, of the text) [77]. The nature of the construction may vary a lot, depending on the corpus and on the editorial project defined by the editor. As Shillingsburg puts it, critical edition can go from “reconstructing now lost texts” or “identifying and correcting errors or stylistic lapses in the text being edited” to “extracting from the plethora of authoritative evidence an intended text not yet realized”.

The aim of this construction is not consensual either. According to Kline, the purpose of the reconstruction is “to establish an authoritative text that does not reflect every element of any single surviving documentary source but, instead, embodies the editor’s critical judgement of what an author’s true intentions were” [106]. One interpretation of this proposition is that the underlying editorial project for critical editions is based upon the hypothesis that the surviving witnesses do not represent the author’s intention, because the intended text was either never made public (e.g. was emended on the proofs, due to censorship⁶) or because it is only accessible through corrupted or faulty copies and editions. The purpose of a critical edition is thus to attempt to recreate what the intended text could possibly have been, based on material evidence, either contained inside the source materials, or based on documents that are not versions of the text but provide information on it (gloss, quotations, paraphrase, letters, etc.) and on the editor’s critical judgement. Recently, some editors, tenants of what is called the New Philology, have raised criticism towards this point of view, stating that it relies on a particular vision of the author, whose intention prevails and shall be reconstructed, inherited from Romanticism, and that leads to consider each variant in the witnesses as “fundamentally faulty” [111]. Indeed, one of the most rigorous critical editing technique, namely, Stemmatology, consists in building a family tree of the witnesses, in a phylogenetic fashion, based on error models; in particular, the principle that a “community of errors implies a community of origin” is

⁶A famous example of that may be found in the ‘proof volume of the *Encyclopédie*’ preserved at the Library of the University of Virginia (ref. Gordon 1751 .D54 Proof vol.). According to the Library’s website (<http://small.library.virginia.edu/collections/featured/224-2/>, accessed on August 5th, 2017), “[t]his volume was apparently made up by one of the editors of the *Encyclopédie*, almost certainly André Le Breton who is known to have effected the unauthorized censorship of many of the articles by Diderot and other contributors. These proofs are in the original settings before Le Breton’s editing and are, therefore, of greatest significance to scholars of the *Encyclopédie* as the only known source of Diderot’s full uncensored texts.”

operated to group witnesses as descending from the same, lost intermediary version, called hyparchetype, and so forth, in order to reconstruct, up to certain hypothesis, the original, archetype text. Yet, the above theory, that relies on the vision of a text being associated to either one author, meaning that any exogeneous intervention has to be considered as a corrupting process, or to one single, original version, is argued not to be transposable to the pre-Romantic era, during which the figure of the author does not translate well, in particular in Medieval times, where scribes, in charge of writing the manuscript copies of a given work, did not necessarily have the role of copists: hence the need, for New Philologists, to give the witnesses of a work their original quality and to consider them as first-order texts, instead of simple witnesses of a lost, hypothetical text [43].

Alongside those passionate debates, that still have very practical, concrete implications, Elena Pierazzo suggests another, pragmatic, historical *raison d'être* for critical editions [135]. She points out that editors have always faced the dilemma between editing “texts or documents (or texts of works vs. texts of documents)”. History shows that most of the time, texts of works were preferred, instead of documentary editions. “This choice has been almost inevitable for works for which many witnesses survive: who indeed, apart from the editor and possibly a couple of other scholars, would be interested in buying and/or consulting seven hundred versions of Dante’s *Commedia*, one for each of the surviving witnesses? Historically, in cases like these, the only sensible solution has been to reconstruct the version that corresponded most with the theoretical orientation of the editor, and to serialize the rejected variant readings in the apparatus. The result is the provision of a clean, reading edition, where the variants are conveniently marginalized at the bottom of the page or at the end of the volume, in the name of ease of reading.” In other terms, critical editions have been favoured – to the point of virtually eclipsing documentary editions out of the scholarly field [135] – due to financial and readability reasons.

Digital Scholarly Editions: Documentary, critical or both (or more)? This last argument seems of particular importance, in order to understand the ongoing evolution of Scholarly Edition in the Digital world. As a corollary of its very reason of existence, as a means to pally the impossibility to exhibit the primary sources used for the edition, “access to the sources has always been one of the biggest limitations offered by traditional critical editions, where only the categories of variants considered relevant by the editor were collected and organized in the *apparatus criticus*” [135]. Yet, since the mid 1990’s, vast digitization campaigns have been launched either by private companies, by public institutions or both jointly⁷, resulting in huge collections of high resolution images of consistent and rare heritage funds, ranging from printed editions, manuscript *avant-textes*, private correspondence, author’s library, etc., now available to the scholar (if not to a larger audience directly⁸). Thus, the access to a graphical representation of the sources is no more a financial issue, as it has been for

⁷As an illustration, in 2008, the Municipal Library of Lyon launched a call for tenders for the digitization of its funds. The campaign was funded by the Library itself, the Ministry of Culture and the National Library. The call was won by Google.

⁸GoogleBooks, in the private world, or Gallica, a state-funded platform, offer a wide range of free, digital heritage resources.

paper-based editions. In this context, it has become possible (and, actually, this is even now a sort of *de facto* rule) to incorporate them: the DSE shall give the reader an access, somehow, to the digital images of the documents upon which the edition is based, that is, to a visually faithful representation of the whole set of witnesses considered for the edition.

As the experience shows – and as was known from traditional facsimile editions, that were of little use among the scholars [147] – images do not make a scholarly edition. Image-based editions lack readability and exploit very little of the perspectives offered by the digital media, in terms of querying and computability. Instead, the current, predominant approach [135], is based on the correspondence between the images, of course, and a highly accurate, at least diplomatic, transcription, retaining as many information about the edited document(s) as relevant. Indeed, this methodology does not solve, or bury, the traditional divergences between the tenants of critical versus documentary editions. From one point of view to the other, the nature of the transcription will differ drastically. For instance, while some may advocate to wipe all exogenous information from the transcription, that is, to transcribe only “what the editor sees as directly attested by the document is to be included in its edition”, that approach would mean, for others, to neglect all the interesting aspects about the work the document witnesses (“what it means, who wrote it, how it was distributed and received, how it is differently expressed”) [147]. Yet, the two positions do not need to exclude one another any more. The necessity of providing a digital version of the material source in a digital project can be seen as a step in a more general type of scholarly editions, which can adapt to the reader, and cover, all-in-one, the range from copy-text to fac-simile editions, through critical edition, on demand, hence overcoming the classical dilemma between documentary, versus critical edition: indeed, “features that were once normalized without mercy to produce reading, critical [...] editions can now be retained and simply switched on and off at leisure to please different audiences, thereby opening the way to new scholarship and readership [...] together with a critical edition, the diplomatic edition of the sources would be offered for readers’ inspection in digital form, where the space constraints that have determined the format and the selectiveness of the printed apparatus no longer apply” [135]. The encoding system defined for the transcription of Dante’s *Commedia*, which aims at retaining both the “text of document” (sticking accurately to what is on the document) and the “text of work” (containing the variants) is an illustration of such an expanded critical approach [26]. Providing an edited, critical text may be of great usefulness “to the general readership”, while “the edition of the manuscript as document can also be justified on scholarly grounds”. This possibility to make a critical edition rest upon a detailed description of the material resources it is based upon, in a documentary fashion, is also advocated by Jerome McGann, tenant of the influential theory of “Social texts editing” [40].

Interestingly, the above methodology, namely, to base the editorial work on a highly skilled transcription of the documents constituting the primary corpus, adapts well to a great range of editorial projects. In particular, it applies well – to the point of being inevitable – to documents for which materiality is of high interest, and that possibly do not even witness any text *per se*. This is often the case for draft materials, or *avant-textes*, as they are called by the French school of genetic criticism.

Genetic criticism. Indeed, after a complete lack of consideration, considerable attention have been given to working (often manuscript) drafts, as records, not of the text of work itself, but of the process of its creation. Very different from the antique and medieval manuscripts traditionally considered by the critical scholars, drafts are most of the time composite documents, whose textual content is not linearly displayed across the media (but rather dispatched into different zones), is of diverse nature (creative text, summaries, commentaries, metatext) and that may even not be the witness of a literary work under construction, but rather the imprint of the activity of a working mind [111]. In this context, the traditional critical approach, as well as a humble documentary edition, are both irrelevant. The first, because the purpose of editing draft materials is not to provide one synthetic text of work, since that work may not exist, and since varying places in a manuscript do not have the same status as variants, among which the editor may choose in order to reconstruct the intention of the author⁹. The second, because draft manuscripts are generally dense, hard to decipher and non-linear, making the understanding of what is going on on the documents hard to follow without sustained attention: more than a sheer (ultra) diplomatic transcription, the editor has to provide the reader with clues about the structure, i.e. about the semiotic nature of each part of the document and, if appropriate, how to navigate through the corpus to get a significant insight of the evolution in the writing.

Printed genetic edition projects, aiming at rendering the uncovered dynamic processes that the documents show, have been attempted, but because the editors had to make use of a complex set of symbols to represent the information of genetic nature, they resulted in “unreadable, unusable, time-consuming, and, in general, deceptive” paper books [135]. On the contrary, the digital media seems to be particularly promising for such editorial projects. Indeed, the combination of dense encodings of the documents and dynamic publishing interfaces enable to bypass the need for any cryptic rendering of the genetic processes at work in the documents: instead, the reader can travel through the documents, aided by the editor, in a most natural way. As Julie André and Elena Pierazzo indicate, a digital edition may even encode “several trips through the intricacy of the manuscript: writing and reading sequences”, the first working as a reconstruction, up to hypothesis, of the different steps of a passage’s writing process, the second indicating the reader the lead to follow in order to be able to read the last writing state. An implementation of that principle was produced, based on a few drafts by Marcel Proust [6]. The edition relies on a documentary, topological transcription of the documents. The reader accesses the transcription by means of an interface that shows the image of the documents, each zone of the image giving access to its transcription. Moreover, the order in which the different zones can be meaningfully read is indicated by means of a timeline functionality, which enables to “replay” the writing process of the page.

Similar considerations directed the ORIGAMI project, dedicated to the *avant-textes* of the *Eloge de Bossuet* by D’Alembert [14]. Three consistent manuscripts of the *Eloge* have been preserved. The purpose of the ORIGAMI edition was to illustrate how D’Alembert, well aware of the dangers of censorship, installed critical and polemic opinions in an institutional text. The polemic writing strategy of D’Alembert, as

⁹Even if the status of variants, as well as the critical approach based solely on the author’s intention, are, as we have seen, debated [43].

shown by Olivier Ferret, can be seen in the way the author gradually, from the early versions of the text to the printed ones, balances the controversial elements between the text of the *Eloge*, which was to be published during his living time, and the posthumous *Notes* [72]. As a demonstration of this claim, the ORIGAMI edition proposed, alongside an access to the hi-resolution pictures of the manuscripts, a transcription of the documents encoding the different layers of correction (insertions, emendations, substitutions, travels) by D’Alembert inside each manuscript version, as well as indications about the places of disruption and of continuity between the successive manuscript versions. The reading interface rendered this transcription in a dynamic mode: it was possible to display the text of the documents layer after layer, from the first version available until the printed version, driving the reader to visually unlayer the successive authorial interventions of D’Alembert first at the local scale, and then from one version to the next. This functionality was meant both as a reading assistance tool, helping the curious reader to decipher the (digitized) source materials, and as the expression of a reading proposition, an hypothesis about the writing process of D’Alembert that the reader was invited to check and, in terms, to confront to his own views – in that, the ORIGAMI approach was reminiscent of – which fills the circle – the traditional, critical editions, result and basis of past and further research.

In the end... Undeniably, the recent years have indicated a bend from in the trajectory of documentary editions, that from a marginal position in the paper-based scholarly world, have come to occupy a central one in DSE, to the point that providing the edited corpus underlying a digital edition has become a *de facto* standard. Yet, as indicated by [135], documentary editing is more a method than a text theory; and while documentary editing has gained favour among digital scholars, as illustrated above, the field of text theory has been more active than ever, animated with strong debates and growing around some new theories. Thus, DSE teams will obviously continue to propose a wide variety of editorial objects, driven by different views on what DSE should be, or by different aims, ambitions and methods.

However, the rise of the documentary methodology as part of almost any DSE project has important consequences, in terms of how digital editors will be working. Indeed, the necessity to provide a digitised, and transcribed, version of the sources within the frame of a DSE imposes the use of an encoding, or structuring language. Indeed, without this need, critical editions could still be done the traditional way, and variants be included in notes, just as in a paper edition – albeit, the resulting text would be displayed on a digital media, equipped with a search engine and some other functionalities – and the same could apply to genetic editions also. But this is not, and as detailed in the short review above, for good, editorial reasons, the way DSE is following. Thus, the resulting “text” of the edition cannot be plain text accompanied by a series of notes. Instead, DSE clearly lean towards the model of a set of images and structured resources (the transcriptions), encoding a lot of material, critical, genetic information about the documents. The limits of the digital enrichment, i.e. the amount of encoded information, based on that model, are virtually indefinite [135]. We have mentioned DSE whose transcriptions blend documentary and critical, topological and genetic views. Those kinds of annotations, and their combination even more so, enable a detailed description of the edited resources, from an internal point

of view. But one interesting prospect of DSE is the possibility to encode, on top of the aforementioned internal characteristics of a document, exogeneous relationships as well, that is, intertextuality. This idea is not new, and has been theorized, and sometimes experimented, under the name of Hypertext [121, 65] or Web of discourse. We leave the final words, in this paragraph, to Hans Walter Gabler [77], with this exhilarating and almost dreamy vision:

“In the Renaissance, when books first became the medium for editions, printers devised breathtaking lay-outs for surrounding texts with commentaries, often in themselves again cross-referenced. In effect, they attempted to construct in print the relationality of what today are called hypertexts. But with books to establish the third, relational, dimension against their material two-dimensionality, has always been a rudimentary gesture, and has always depended on involving and stimulating the reader’s imagination and memory. For editions existing electronically, in contrast, the relational dimension is a given of the medium, and complex relationalities may be encoded for them into the digital infrastructure itself.”

1.2 Constructing Digital Scholarly Editions: A Generic Approach

Our PhD work originates in an opportunity, supported by the Rhône-Alpes region, in France, to study how digital scholarly editions are manufactured, and then how, from an experimental point of view, the manufacturing process of DSE could be eased, or assisted, or improved. Importantly, the challenge was to adopt, for this study, a generic approach, that is, to make sure that the proposed solution be not specific of one, or even a few DSE projects, but embrace DSE in general. Yet this study is grounded on four digital scholarly projects, that accepted to participate to the intellectual adventure this PhD work represented and to serve as a bank of examples: the edition of the documentation Gustave Flaubert gathered for his unfinished novel *Bouvard et Pécuchet*¹⁰, the exploratory analysis of philosopher Jean-Toussaint Desanti’s archive¹¹, the double publication (printed and online) of Stendhal’s *Journaux et papiers*¹² and the critical edition of the Diderot and D’Alembert’s *Encyclopédie*¹³.

The four projects differ in many aspects. In 2013, at the beginning of the PhD work, three of them had already started, and were even publishing their editorial achievements, while the work on the *Encyclopédie* was in its most early stages. The nature of the editorial aim, for each project, was different. The edition of the preparatory files of *Bouvard et Pécuchet*, which is mainly the work of Stéphanie Dord-Crouslé, aims at providing the reader, who is understood as a ‘user/researcher’ as well, with a range of transcriptions (from ultra-diplomatic to normalized), so that she can then build herself a hypothesis about how the materials gathered by Flaubert could have

¹⁰<http://www.dossiers-faubert.fr/>

¹¹<http://institutdesanti.ens-lyon.fr/>

¹²<http://manuscrits-de-stendhal.org/>

¹³<http://enccre.academie-sciences.fr/>

been assembled into a volume. The approach is thus essentially documentary. The edition of Jean-Toussain Desanti also operates a documentary approach, doubled with an archival aim. The first task was to catalogue the whole archive, and to build an index of names and concepts, describing the documents at a macroscopic level. In parallel, the documents had to be transcribed, in order to allow the identification of unedited works by the author, hidden inside the archive, and also to support a genetic reading of Desanti's work, centered on "his working methods, his focuses, his influences and their evolution through time"¹⁴. The project has been based on a succession of small, specialized teams, sometimes reduced to one researcher. The edition of Stendhal's *Journaux et papiers* is a twofold edition: providing the reader access to the manuscripts preserved at the Municipal Library of Grenoble, by means of a documentary archive freely available online, complemented by a critical, printed edition (see [168], published in 2013). The editorial team, directed by Cécile Meynard and Thomas Lebarbé, gathered up to about 20 multidisciplinary members. Eventually, the name of the ENCCRE project, which is an acronym for "Edition numérique collaborative et critique de l'*Encyclopédie*"¹⁵, is self explanatory. It is the only, of all the DSE projects here, to specifically claim for a collaborative organisation. It relies on a large and multidisciplinary network of scholars (about 100 people) that spreads across Europe, Japan and the United States.

Those projects, because of their diversity, provide a valuable set of real-life experience of DSE. In the following, we propose a general study of how DSE are manufactured. As a disclaimer, this study does not aim at covering the whole range of questions digital scholars face during the construction of their DSE. In particular, some practical, and extremely important decisions, regarding data preservation and interoperability, that are at the very heart of Digital Humanities today [136], will voluntarily not be considered. Those are well-identified, debated problems, that are managed technically and politically at the institutional level. Instead, grounded on the historical experience of the four edition projects above, we propose an abstract description of DSE construction. This description, that aims at being as general (or generic) as possible, will uncover some crucial difficulties and challenges that all four of the DSE projects have faced. Our conclusions can be summed-up as follows:

- From a certain point of view, the activity of DSE manufacturing can be widely described as a modelling and data structuring activity. The centrality of modeling, and the connection with data structuring, has been identified elsewhere [119, 136, 40, 163, 102]. Practically speaking, editors will, based on their editorial policy and on a model of the edition, and of the edited corpus, define a data structure and instantiate it.
- The definition of the data structure is thus a process, or should we say an editorial process that, as such, has raised little interest so far. While heavy with consequences for the DSE, it happens that the activity of defining a data structure is often centralised, even in collaborative settings, and thus editors may be deprived from participating in that process, while they will have to instantiate its outcome (i.e. the data structure itself).

¹⁴<http://archive.desanti.huma-num.fr/desanti/a-propos>. Accessed on August 8th, 2017.

¹⁵Id est, quite transparently, "Digital, Collaborative Critical Edition of the *Encyclopédie*".

- Since the definition of the data structure is grounded on a model, and that “modeling succeeds intellectually when it results in failure” [119], it follows that data structures are susceptible of changing in the course of a DSE project. Data structure evolution in the context of DSE have also raised little attention, to the best of our knowledge.

As one can see, the above points do not describe technical difficulties, but difficulties of an editorial nature – they are not technical first, even if, as we will see, they do imply several profound research questions to the computer scientist...

1.2.1 A Generic Approach of DSE

As we have seen, Digital Scholarly Editions cover a wide spectrum of editorial projects and form a complete galaxy of diverse objects, even if they tend to revolve around a common methodology, based on documents transcription and enrichment. Unspectacular as it may seem at first glance, this methodological similarity across very different projects has important consequences, and can be exploited to provide a certainly rough, but revealing, generic model of the manufacture of DSE.

1.2.1.1 A Starting Model for DSE Manufacture

As a starting point, let us consider a first, rough and incomplete model of how DSE are built. This elementary model, that will be refined in the following, is given in Figure 1.1. According to this model, the DSE results from interactions between subjects (the editor or the editing team) and objects of diverse nature:

- An **editorial project**. It is defined by the editors. It sets the editorial principles according to which the DSE has to be made. It covers a wide range of decisions: “the choice of who the edition should be aimed at; whether it should present an edited text; if so, on what principles should that text be edited; what conventions of spelling, punctuation and presentation should be followed” [148], as well as, in a context involving documentary editing, what characteristics of the documents are relevant.
The editorial project can be tacit or explicit. Most of the time, it is summed up in a text written in natural language that will be, at least partly, available to the final reader of the edition.
- The **data structures**. Regardless of the technological aspects of the encoding (i.e. what language is used for the transcriptions, etc.), in the context of DSE, the data structure is a model for the edition, respecting a certain formalism, so that this model is interpretable by the machine [164]. Those familiar with the way data structures are defined in XML or in TEI, that is, by means of a schema, are well aware of that. Indeed, as suggested by Willard McCarty, *model* is a double-sided concept, referring both to “a representation of something for purposes of study, or a design for realizing something new”. He distinguishes between “models of” and “models for”, to enlighten the above distinction. Data structures, indeed, are “models of”. Let us stick to the example of XML-based

editions. A schema for XML contains the definition of the elements that will be instantiated (i.e. the types of the objects that will be identified in the sources) and of the hierarchy those elements will constitute in the annotation. Thus, defining the schema for an edition demands to have identified the objects of interest in the sources, that will, in the transcription, be represented by an element, as well as the way those objects of interest relate one to the other, from a hierarchical point of view. In other terms, the editor will only be able to define the schema for the transcriptions once she has got enough familiarity with the sources and the project so as to have developed a good mental image, or a good model, of the characteristics of the corpus to be edited and of the editorial enrichments intended for it¹⁶.

At the same time, a data structure is generally not purely descriptive. Data structures defined as a preliminary for the editing work, as it is commonly done, play a strong, prescriptive role as well, since all the data generated afterwards by the editors will *have to* be conform to that data structure. Data structures work like a guide for the editor, but also like a censor: only the information planned in the model can be expressed, as planned in the model – an XML file containing an element not defined in the schema, for instance, being necessarily considered invalid.

The data structure works thus as a definition of the language of the annotation: it defines the conceptual vocabulary and the grammar that will be available for the editor to express the annotation.

- The **digital corpus**. This is what the editors construct while editing. In the first steps of the DSE construction, the digital corpus corresponds to the unedited files: native-digital sources, images of the documents, raw transcription or text issued from the OCR... Later, the digital corpus will be made out of the enriched data. In case a data structure was defined, the corpus will have to respect that structure, in the end.
- The **material corpus** is the set of material objects that need to be digitized.
- An **editing interface**. It is by means of that interface that the editor will be able to define/construct the machine interpretable files corresponding to the edition: the data structure, if one is defined, and the digital corpus.

This basic model of the manufacturing environment for DSE provides a set of six abstract subjects/objects for which we can now try to find a generic incarnation, that, among the possible shapes, subsumes (or generalises) all the others.

1.2.1.2 A Composite Approach of Genericness

In our claim to propose a generic approach of the DSE manufacture, we understand “generic” by “lacking specificity”¹⁷: a minima, “generic” thus means *versatile*; a maxima, *universal*. We follow the principle that a system (composite, by nature), as the

¹⁶Moreover, from a computer science point of view, data structures are models of the data in that they can be exploited to formulate, or optimize, complex queries on the data.

¹⁷See www.thefreedictionary.com/Generic. Accessed on August 9th, 2017. The same definition applies in French, see atilf.atilf.fr/.

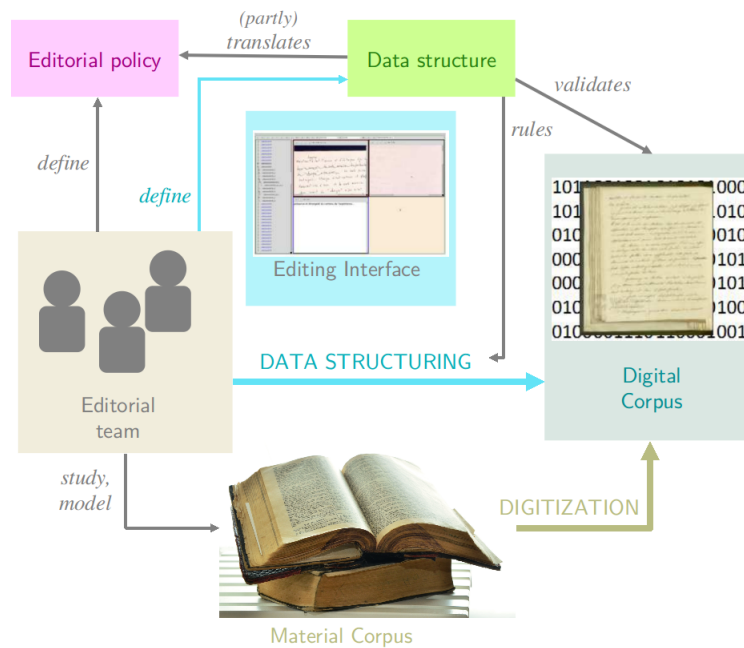


Figure 1.1: Elementary model for DSE construction.

one we have modelled in the previous paragraph, cannot pretend to be versatile if one of its parts suffers from a too specific definition. We now try to consider each element of the above model for DSE construction, and to provide them with the definition of what might be their most generic realisation.

- **Editorial team.** We consider, as the most general case of editorial team, a collaborative, distributed, multidisciplinary team of scholars that are not familiar with computer science. This subsumes editors working alone, teams working at the same physical place and/or benefiting from the possibility of meeting regularly, etc. It is also open to the eventuality of there being some kind of turnover, or evolution, in the composition of the team.
- **Editorial policy.** The editorial policy is connected both to the editorial project the editorial team forms and the sources to be edited. It would be presumptuous to summarize all the possible views one can have on any primary sources. Still, to be as general as possible, we can say that one DSE project may, as has been illustrated in the paragraph 1.1, support several editorial paradigms at the same time (genetic and documentary, critical and, say, linguistic, etc.). In other words, in an editorial system supporting the most generic kind of editorial policy, the editors must not be restricted to express annotations conform to one paradigm only: double, or multiple-paradigm annotation must be possible. Even more so, it shall be possible to decide, after the project has started, to

change the editorial policy, for instance, but not solely, by going from a single-paradigm to a multiple-paradigm edition. Thus we articulate genericness, regarding the editorial policy, with two notions: *multiplicity* and *dynamism*, as follows:

1. at any moment, several editorial views may be expressible;
 2. the editorial policy may evolve through time, under the spur of the editors.
- **Data structure.** Since the data structure is a partial translation of the editorial policy:
 1. the data structure shall support multiple annotation of the digital resources;
 2. the data structure shall not be definitive. It must be a first-order object that, as such, the editors shall be able to amend, in the course of the DSE manufacture.
 - **Digital corpus.** The digital corpus may be a set of resources of diverse nature: image, necessarily, but also annotated text, audio and video resources, native-digital objects... The digital corpus is by nature dynamic, since the editors, by encoding information in it, add secondary content to it. The primary content as well may change during the editing work.
 - **Editing interface.** Generic editorial interfaces have been proposed elsewhere. Such interfaces aim at covering the whole range of editorial activities necessary to the construction of the edition, from the data structure definition until the parameterization of the published materials. The Glozz platform, primarily dedicated to linguistic annotation, exemplifies such interfaces [183]. In particular, it enables to tune a “data metamodel” (i.e. to define the data structure) according to several linguistic paradigms; it provides user-friendly tools to instantiate on the corpus those different paradigms simultaneously; it enables to query the resulting digital corpus and to generate files in different formats (XML, TEI, txt...). Modular interfaces, designed by the user herself from a library of pre-existing functional bricks, can be seen as another way to achieve the same goal [11]. It shall also support collaborative work, as hinted by the above point on the nature of editorial teams.
 - **Material corpus.** Similarly to digital corpus, the material corpus may be multimedia, and may change during the lifetime of the edition.

1.2.1.3 Conclusion of this Paragraph

Genericness is defined as the lack of specificity. What the above suggests is that a non-specific digital edition system shall be adaptative, i.e. shall allow the editors to define their own language of annotation freely, and to instantiate it on the data.

We also get the hint that non-specificity shall be understood in dynamic terms: a generic edition system shall support both stable editorial projects, that keep the same editorial team and policy throughout their lifetime, and evolving projects, for which the corpora, the team and the policy, may change any time.

We will thus now consider how the combination of those two characteristics onto which genericness is grounded (adaptation and evolution¹⁸) impact the main activity in the digital edition routine, namely, data structuring (and its corollary, data structure definition). The questions we will try to answer, on a theoretical (not technical yet) level, are the following:

- What does the activity of data structuring consist in? How may an editor want to make a data structure evolve and what difficulties does it raise?
- How does the above translate in the context of collaborative work?

1.2.2 Data Structuring Models

By structuring, we refer to the whole process consisting, on the one hand, in the definition of the types representing some characteristics of the data and of the relationships between those types and, on the other hand, in the instantiation of those types in the data, that is, the annotation itself.

It is worth noting that the first activity is, sometimes, indissociable from the second, namely in the editorial systems that rely upon the notion of implicit data structuring: in this case, the data is not structured according to any pre-existing, formally defined structure, and the data structure is the structure that can be inferred from the data. Explicit structuring, in which data structures pre-exist to the data, is the alternative. We question those two kinds of data structuring for DSE in the following paragraph.

1.2.2.1 A Panorama of Data Structuring Paradigms

Implicit structuring. As stated above, defining the data structure for an edition demands to have a clear model in mind, and to be able to formalize it. Yet, some projects might not either be in capacity of, or want to, define such a model *a priori* of the editing work. As pointed out by [156], the formalistic burden this represents can be an obstacle for the activity of collaborative groups, or of scholars, that can (and at the time the article was written, certainly were in a vast majority) unfamiliar with the activity of formal modelling. Following those considerations, several propositions aiming at relieving the scholars with abstract structuring have been proposed.

CritSpace is an interesting such proposition. It takes the shape of an interface for the exploration and the organisation of multimedia, heritage corpora. Thus, structuring is at the very center of the aims of the interface. The interface, from a user point of view, relies upon the ergonomic paradigm of “spatial hypertexts”. The way structuring is done in CritSpace is the following:

- The user is provided with graphical representations of the documents that constitute the corpus to investigate. The user can then display those representations on a working surface (a window, actually) and position precisely those representations on this surface. It is then the spatial proximity of two documents’

¹⁸Interestingly, this abstract analysis, that dates back from 2014, at the start of this PhD work, is shared, almost word for word, by [103], that gives an account of how, practically speaking, genericity was achieved in the DigitalHarlem project.

representation that encode the existence, and the tightness, of a semantic relationship between the two corresponding documents.

- It is then possible to write textual notes (in a post-it manner) to comment on the displayed resources.
- It is also possible to “synchronize” resources, in a very specific sense: when one resource from a set of synchronized resources is manipulated in the interface, the others of the set are highlighted, so that the user be informed of the existence of a particular relationship between the documents those resources represent.

Figure 1.2 is an illustration of the CritSpace working surface.

This structuring paradigm is clearly dedicated to the exploration of a documentary corpus – in this regard, the authors do not mention any means to export the result of the structuring by CritSpace. It is also semantically quite limited, since it is not possible to type either the resources nor their content, nor the nature of their relationship. Other tools, by contrast, are grounded on the notion that the abstract quality of structuring needs not be erased, but facilitated [156]. A quite recent example can be found in Analyst’s Workspace [7]. In this approach, structuring is incremental: first, it is quite similar to the way it works in CritSpace, in the sense that the (textual) resources are classified according to their spatial display on a working surface, without any typing of any sort. Then, the user can formally define new objects called “entities”, which are then associated with a zone of the working surface and, by corollary, the resources that occur on this zone, hence working as a sort of typing of the network they form. However, no export is described by the authors neither.

Let us denote that none of the two systems presented here make it possible neither to infer any structural information from the structured data resulting from their use, nor to exploit that structure for querying purpose. By contrast, one can think of the proposition of Dataguides and Graph schemas, two complementary tools proposed to infer, or extract, a descriptive structure from data that possess, at least partly, some structure, but that was generated without the use of any kind of schema (or predefined data structure), or that were aggregated from diverse sources of information [3].

Explicit structuring As it happens, the above examples are not meant to be used for editorial purpose – unless in a preliminary, research-oriented phase of work that is not specific of edition. As it happens also, examples of editing systems that would rest upon an implicit-structuring paradigm lack, to our knowledge. Instead, it seems that scholars, in the enthusiast atmosphere of Digital Humanities, have gained the competence of thinking in terms of abstract models and data structure, to the point that some tools, like the aforementioned Glozz platform [183], not only give the scholars the possibility to define a data structure themselves, but also to annotate that structure as a first-order object. Additional evidence comes from the fact Digital Humanities literature does not hesitate in discussing advanced problems of modelling and data structuring [58, 68, 177, 60, 103].

Thus, it appears that the concerns expressed in [156] are not justified anymore, or at least, does not apply to DSE editors, that are well-aware of data structuring problems and in capacity of dealing with them. Explicit structuring can be divided into two

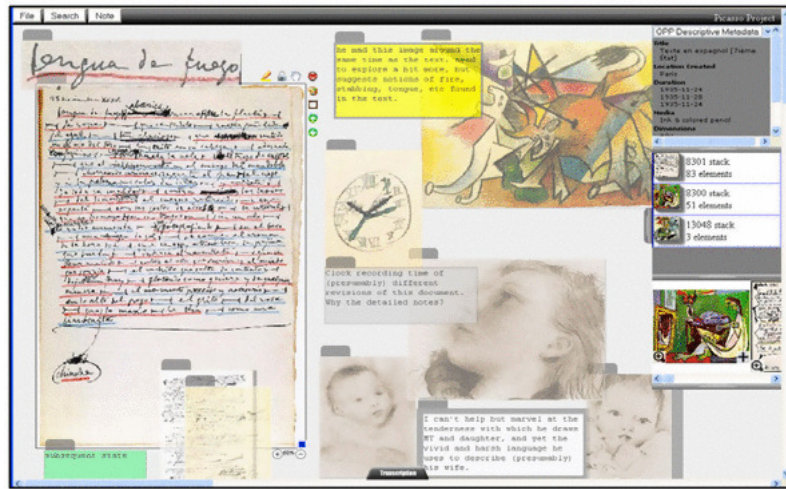


Figure 1.2: The working surface of CritSpace. The information that two documents share close relationship is indicated by the spatial closeness of the graphical representation of those documents.

cases: in the first one, defining data structures is equivalent to defining types, that can then be instantiated inside the data, without there being a necessity that the data structure be instantiated, and the other, in which the data structure is designed as a prescription over the structure of the data, that must conform to it in order to be considered valid. The two philosophies can be illustrated as follows:

- In the Glozz platform, the user is given the means to define several data structures each supporting a certain annotation paradigm. Those data structures are quite simple. They derive from a metamodel in which “segments” (types used to characterize segments of text) can be connected together by “relations” (typed associations between segments); sets of relations, of segments connected by relations, can then be typed, forming what we might call “typed substructures”. The data structure, as a whole, can thus be quite complex, containing intricately connected segments, grouped into several, overlapping typed substructures. It is then possible to instantiate each kind of segment into the data, and then to connect the instances of segments by relations as defined in the data structure. Still, the data structure, in this explicit structuring paradigm, only works as a library of types (for segments, relations and substructures). In particular, even when a segment is part of a typed substructure, it can be instantiated alone, without it being necessary to instantiate the substructure as a whole.
- On the contrary, data structures are designed to have a far more prescriptive role for the well-known and widely used annotation (meta)language XML [28] is. In XML, the explicitly defined data structures are called ‘schemas’. There are several schema languages for XML, from DTDs to XSD and RELAX NG

[126]. The XML model relies upon the notion of element, which correspond to a typed ranges of an XML document that can be affected attribute values (or metadata). XML schemas also rely upon the notion of element type, or type of element: they contain the definition, not only types of elements, but of a family of hierarchies of element types, that shall reflect the model of the edition the editors have made – with the constraint that this model shall be hierarchical. Schemas are then used to check if a given XML document has an appropriate structure – it is said to be ‘valid’ if so –, which happens if and only if the set of all the elements contained in the XML document form a hierarchy that matches one of the hierarchies of element types defined by the schema.

The philosophy here is thus quite different from the one in Glozz, which results in important practical differences. Compared to the ‘library of types’ proposed by Glozz, in which one can pick the types she wants to instantiate in the data, XML schemas describe the structure of documents as a whole: if one wants to instantiate a given element type defined in the schema, then, she has to instantiate a whole hierarchy of element types containing the element type. While Glozz philosophy can be described as ‘pick whatever you want’, XML schemas are in a ‘take all or leave all’ fashion.

This last remark means that schemas play a prescriptive role Glozz data structures do not. If a schema is given, then the structure of the documents one may produce, and that need to be validated by the schema, is limited. This aspect of the schemas is fundamental. It is actually one of their essential reason of being: considering a situation in which several co-acting people are supposed to write XML documents for a common use, or for exchange purpose, then schemas, if they do limit the expressiveness of the the co-actors, ensure that they write documents that share the same structure. Moreover, in a DSE construction setting, schemas can be exploited as a means to help the scholar to build the annotation: XML editors softwares like Oxygen, provide the user with ‘content assist’ features, that offer suggestions for completion, according to the context and to a predefined schema. This is indeed a very useful feature, because it lightens the burden of text encoding and, as a consequence, helps commitment into annotation by the experts.

1.2.2.2 Making the Structure Evolve

As key factors of genericity for the construction of DSE, we have identified the two following points:

1. the data structure must be adaptative, that is, it must support a vast array of editorial policies;
2. the data structure must be evolutive, that is, editors shall be able to amend it during the course of the manufacturing of the edition, so as to meet the needs of DSE projects that change, purposely or by accident, in their lifetime.

The first of the two requirements is nothing original. Indeed, all the aforementioned propositions for data structuring meet it, to a variable extent. In particular, the letter ‘X’ in XML precisely means that XML is ‘extensible’, in the sense that it does not

rely upon a closed, predefined set of elements: any user can define her own. Yet, we will see in Part II that the range of annotations allowed by XML is limited – and is, most importantly, limiting, in practice.

The second requirement is more ambitious, and has raised far less attention. In the following, we propose an analysis of how and why data structures may evolve, in a DSE project. For that purpose, we ground our analysis on the history of the four DSE projects associated to this work.

Evolving Data structures: an example . This first example is taken from the early history of the ENCCRE project. The project aims at providing the first digital critical edition of the *Encyclopédie* of Diderot and D’Alembert. To achieve that goal, two fundamental principles were adopted when the project was launched in 2012: first, because of its critical nature, the edition had to reflect the state of knowledge on the work; second, the edition had to rely upon a descriptive and prescriptive data structure, accurate enough to enable to encode that knowledge.

Regarding the material sources considered for the edition, a copy-text approach was chosen, based on a particular exemplary of the work, held by the Mazarine Library¹⁹, in Paris, which is from the first printed batch of the first Parisian edition.

The first task the editorial board assigned itself was then to design a data structure that would reflect both the intrinsic characteristics of the document and enable to express the result of the studies that have been conducted on the text of the *Encyclopédie*. Regarding the intrinsic characteristics of the document, since the *Encyclopédie*, roughly speaking, takes the shape of a dictionary, when looking at a page, one may perceive a certain (and, actually, quite fuzzy) structure. The *volumes* are divided into *articles*, that start by a *vedette*, term or series of terms that announce the subject of the article, etc. The data structure thus had to enable the identification of the *Articles*, *Vedettes* and *Volumes*, for instance, in the transcription of the document.

Regarding the studies on the *Encyclopédie*, of particular interest are those that aim at identifying the authors of the articles the *Encyclopédie* contains. Indeed, while some articles are signed, by means of symbols whose signification is given in particular in the *Discours préliminaire* [110], others are not signed. Sometimes, several contributors provided the different parts an article is made of. Attributing studies thus aim at identifying the authors of the different passages of an article, both signed and unsigned. The data structure thus had to enable the identification of *Attributed passages*, the *Signatures* and the identified *Collaborators*.

The first model, or data structure for the articles²⁰ that was intended by the editorial board, that we may call *Alpha*, was the following: An *Article* contains a *Vedette*, *Signatures* and possibly *Attributed passages*. An *Attributed passage* may relate to a *Signature*. A *Signature* is related to one and only one *Contributor*. The data structure is represented²¹ on Figure 1.3. Let us consider that this data structure is instantiated on a digital corpus *A* restricted to the transcription of one article of the *Encyclopédie*, “Jet d’eau”. According to that data structure, attribution was to be encoded as follows:

¹⁹Item 2° 3442, Bibliothèque Mazarine, Paris.

²⁰... presented here with many omissions, for the sake of simplicity.

²¹At that state of their work, the editors of the ENCCRE project had not chosen a particular language for the text encoding. The data structure was represented in a UML-like graphical form.

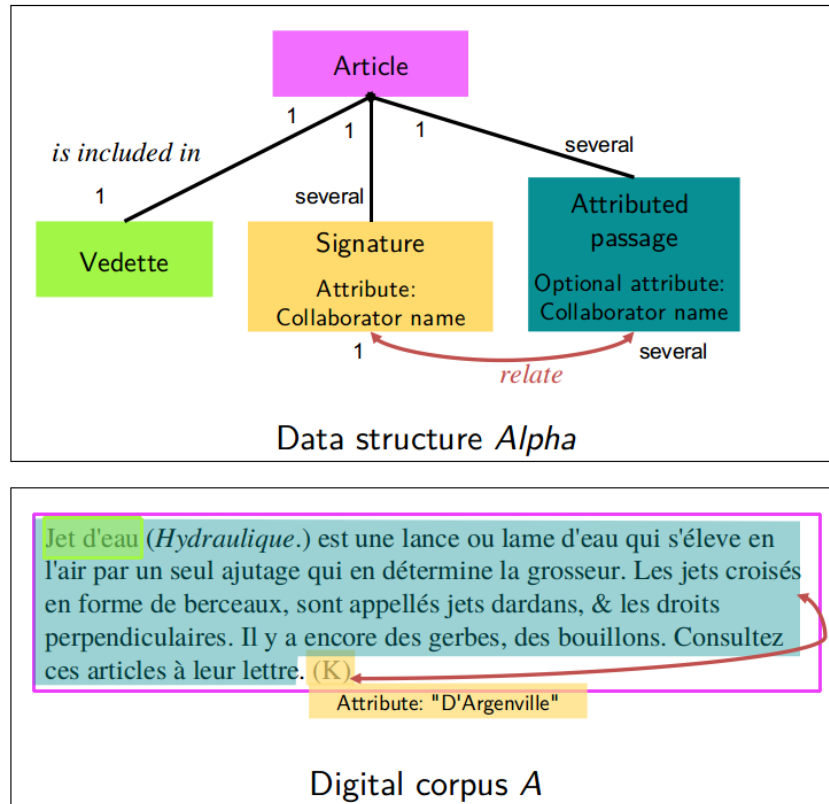


Figure 1.3: Representation of the data structure Alpha, instantiated on the digital corpus A.

1. In case of signed passages, the symbolic *Signature* is identified in the document, the attribute value of the *Signature* set to the collaborator name that corresponds to the symbol, according to the *Discours préliminaire*; the *Attributed passage* is delimited in the document; a link is created to relate the *Signature* to the *Attributed passage*.
2. In case of unsigned passages, only the *Attributed passage* is identified, its attribute value set to the collaborator name uncovered by the editor.

This strategy works well on the digital corpus A, made out of one *Article*, made of one *Attributed passage* related to a *Signature*. It would also work well for unsigned passages.

However, a closer scrutiny at the secondary literature on the *Encyclopédie* provided a counter-example that is not handled by the data structure Alpha. Indeed, the article “Allées de jardin” is made out of two signed halves. The first, signed (K), for

D’Argenville, is indeed from the hand of D’Argenville – this situation is similar, in all aspects, to the one found in the digital corpus *A*. Yet the second half, signed with Diderot’s star (*), has been attributed to... D’Alembert by Fabrice Ferlin [71]. This finding introduced the idea that some symbolic signatures printed in the *Encyclopédie* may be fallacious. Hence the need for another data structure, *Bêta*, represented on Figure 1.4. That new data structure contains a new object, named *Collaborator*, and a new kind of relation, that can bear a note written by the editor. According to that structure, attribution works by relating the *Attributed passage* to the *Signature*, for signed passages, and the *Signature* to a *Collaborator*, by means of an annotated relation: the note attached to the relation will be the place where to justify the difference between the symbolical signature on the document and the real author of the passage.

In this case, in order to take into account the secondary literature on the primary corpus, the editorial board must have redefined the data structure initially chosen. It is possible to be more accurate upon the nature of that “redefinition”. The shift from Alpha to Bêta did not imply a complete change of the data structure: most data types (*Article*, *Vedette*, *Attributed passage*, *Signature*) remain. Qualitatively, the shift from Alpha to Beta can be decomposed as the dropping of the attributes in *Signature* and *Attributed passage*, by the insertion of a new type *Collaborator*, and by the definition of a new kind of annotated relation. Thus, *Bêta* does not simply replace *Alpha*: most importantly, ***Bêta amends, rectifies, Alpha.*** Moreover, in this case, the corrected structure does not subsume the amended one. The situation implied by this amendment is then the following:

$$\begin{aligned} \text{Digital corpus:} & \quad A \cup B \\ \text{Data structures:} & \quad \textit{Alpha} \perp \textit{Bêta} \end{aligned}$$

In this case, we then face a situation in which the digital corpus is divided into two parts ($A \cup B$), each instantiating two incompatible data structures. The data structure amendment thus generates conflict inside the structured data. Such conflicts shall appear whenever the amended data structure does not subsume the previous one, that is, when the nature of the amendment was not solely addition. We provide an exploratory typology of data structures transformations that shall be useful, and that imply conflicts, right below. But first, we give a few examples, taken from the real-life of the other three editorial projects, that advocate for the need, in a tool dedicated to the construction of DSE, to anticipate data structure evolution, i.e. to assist it, as well as the subsequent data update.

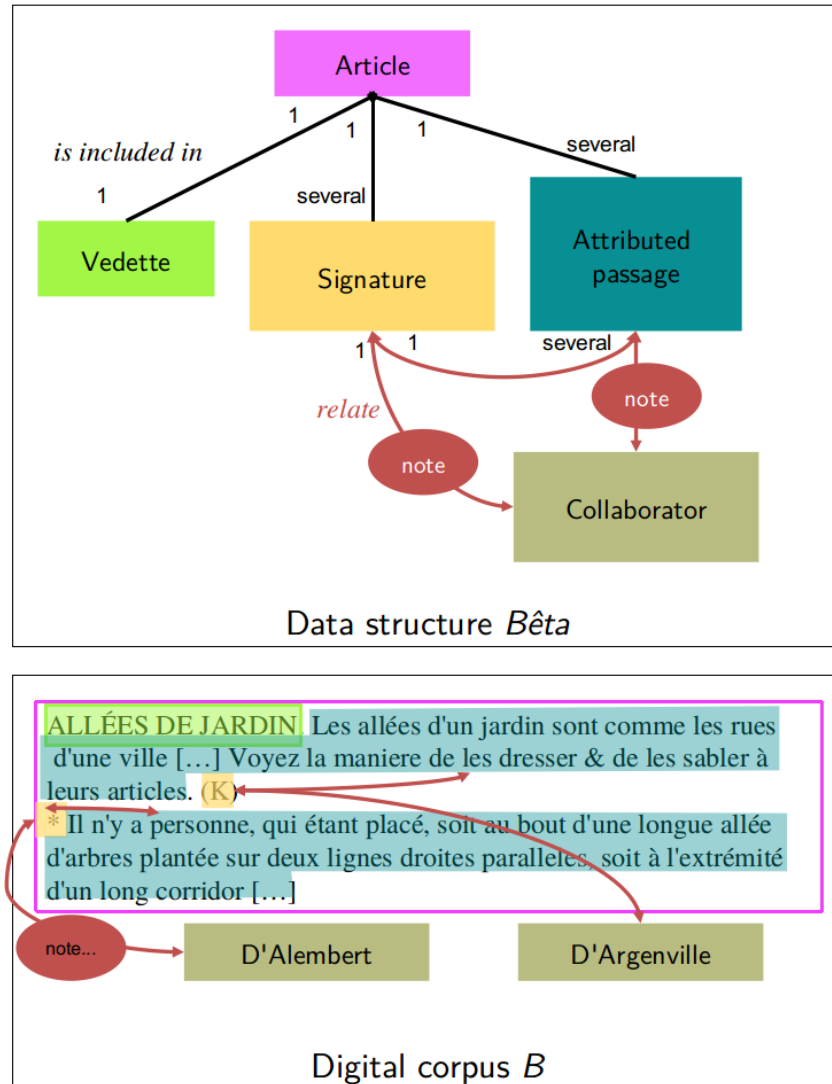


Figure 1.4: Representation of the data structure *Bêta*, instantiated on the digital corpus *B*.

Examples of Data structure evolution

1. **Manuscripts de Stendhal** A very precious resource, attesting of the way data structures do change along the construction process of a DSE, is the Wiki²² created to support the team work for the edition of Stendhal’s manuscripts. It provides us with a view of the history of the DTD modifications from April 2007 to February 2011. The modifications are of several orders: addition of an element type, deletion of an element type, addition of an attribute, change of the list of fixed values for an attribute, specialization of an element. The data update, consecutive to those modifications, have been done by hand mainly, or at times, by means of ad-hoc scripts.
2. **Desanti Archive** While the encoding of the documents of the Desanti Archive had already started, new material sources were communicated to the ENS Lyon, where the archive is preserved. Those new elements were audiotapes of lectures given by the author. As such, they were included in the primary corpus of the project. Those resources have strong connexions with the text material of the archive known so far. Yet transcribing those demanded the data structure to evolve, so that the encoding of audio files, and of the correspondence between the discourse and the textual content of some textual documents, be at hand. By lack of editorial members, the data structure has not been adapted to achieve the whole perspectives opened by those new materials yet, and remains a project.
3. **Dossiers Flaubert** The data structure validating the transcriptions of the Dossier enables the identification, in the pages, of fragments, material and logical units of the *Dossiers* [64]. The data structure enables the editor to identify the title, the identity of the person who copied the fragment and the bibliographical references contained in the fragment. Recently, after the instantiation of that data structure on most of the digital corpus, the editor found a counter example of a fragment that possesses not only one, but several titles. By lack of technical staff members, the data structure has not been adapted to achieve the whole perspectives opened by those new materials yet, and remains a project.

A typology of useful data structure amendments . Based on the general model of a data structure as the formal document defining both the *types*, or characteristics that can be attributed to parts of the digital corpus, resulting in objects generally named as ‘elements’ [28], ‘range’ [175] or ‘segments’ [183], of the *attributes* describing more finely the instances of those objects, and of the *relationships* those objects may have, we finish this part on data structuring and data structure evolution by sketching a small catalogue of the data structure amendments that, in the light of the above and by anticipation, may be useful. For more clarity, we propose to illustrate the different cases by toy examples based on the data structure of the Articles of the *Encyclopédie* already discussed. The typology can be found in Table 1.1

²²Available at <http://stendhal.msh-alpes.fr/index.php?n=XML.ToDo>. Accessed on August 10th, 2017.

	Kind of amendment	Example	Resulting data conflicts
1.	Specialisation or generalisation	In the <i>Encyclopédie</i> , the ‘vedettes’, that give the subject of the article they are positioned at the start of, can be of two kinds: ‘adresses’ when in capital letters, and ‘entrées’ when in small capital letters. Hence the general type <i>Vedette</i> , in the data structure, can be specialised into two types (for which <i>Vedette</i> is a generalisation): <i>Adresse</i> and <i>Entrée</i> .	It can be decided that after the specialisation of <i>Vedettes</i> , only <i>Adresses</i> and <i>Entrées</i> can be identified in the corpus – no <i>Vedettes</i> anymore. In this case, the data preceding the amendment is outdated, since they contain instances of the <i>Vedette</i> type.
2.	Deletion or addition	The identification of some characteristics, or the presence of some attributes for some elements, or some relationships, once judged relevant, is not any more. Or conversely, some characteristics of the corpus lack in the data structure.	Deletion is not quite problematic: the deleted types are simply erased from the data. Added types, on the contrary, if they are compulsory, will be missing in the data preceding the amendment.
3.	Substitution	Instead of modelling <i>Adresses</i> and <i>Entrées</i> as special cases of <i>Vedettes</i> (see line 1 in this Table), the editors may prefer to assess that there are only <i>Adresses</i> in the <i>Encyclopédie</i> , with a special case for <i>Entrées</i> . The type <i>Vedette</i> in the initial data structure is thus replaced by a type <i>Adresse</i> , with a specialisation type <i>Entrée</i> .	In this example, all the instances of the type <i>Vedette</i> must be replaced either by <i>Adresse</i> , or <i>Entrée</i> , accordingly.
4.	Addition or deletion or substitution of values in a fixed set of values (for attributes)	Same as above	Only substitution is problematic. See line 3 above.

Table 1.1: Typology of the different kinds of data structure amendments, and the resulting data conflicts.

1.2.2.3 Intermediate Summary

In the light of real-life examples, it appears that editors are closely interacting with the data structure: they define it, but also, at times, they need to redefine, or amend it. Indeed, the data structure is a means by which editors have to translate the model of the edition that they have designed while defining the editorial policy into a machine-interpretable shape that will, in turn, guarantee that that model is respected by the transcription: in other words, the data structure defines the language of the annotation, that is, the expressive power that is in the hands of the editors. For that very reason, it looks like a reasonable objective to attempt to give the editors control over the data structure, since this would mean giving them the means to shape their own expressivity, in the frame of the DSE.

Those conclusions on data structuring, that were drawn without considering the working context in which structuring happens, must now be confronted with the actual working organisation we consider as the most generic, that is, collaboration.

1.2.3 Collaborative Data Structuring in DSE Projects

So far, we have considered data structuring as a disembodied activity. Data needed to be encoded according to a data structure, that may change – and we studied to what extent that structure shall change.

This approach of structuring would have been similar if we had considered the way structuring is done in a single-editor setting. Indeed, an editor alone can define quite easily the information she wants to encode; it thus makes sense to provide her immediately with tools that enable her to tune her digital expressiveness, that is, to define and amend the underlying data structure. Yet, as we have seen, the case of one editor working alone is one among a vast diversity of working organisations, the most general of which being represented by collaborative, distributed, multidisciplinary edition groups. Then, does the above definition of data structuring apply to this most general context? What does it mean then for a group of editors to decide on the expressiveness they need? Is it relevant to leave the editors tamper with the data structure that gives shape to their collective product, in an immediate, unsupervised way?

To answer those questions, we first provide a definition of a collaborative, distributed, multidisciplinary editing team, before showing that the change of scale implies a new model for the activity of the editors, and a finer definition for “evolutive data structures”.

1.2.3.1 “Collaborative, Multidisciplinary, Distributed Editorial Team”: A Working Configuration with Paradoxical Needs

Lexicographical precisions. We propose a definition of “Collaborative, multidisciplinary, distributed editorial team” hereafter.

1. **Collaborative team.** There is no consensus upon what the expression “collaborative work” refers to. “Collaborative” does not appear as a very specific term in the dictionaries. *FreeDictionary*²³ states that it means “to work to-

²³thefreedictionary.com. Accessed on August 10th, 2017.

gether” and is synonym with “cooperative”. The same, general definition is given in the Cambridge Dictionary, the Merriam-Webster and the Harper Collins²⁴. While Computer Science literature also tends not to distinguish between the two terms²⁵, some French resources seek to make a clear distinction between “collaboration” and “coopération”. According to [90], the most central point is that in a *cooperative* setting, the coactors split the general task to achieve into several smaller tasks, whose execution is then distributed among the team (following the principle of the Division of work); in a *collaborative* setting, on the contrary, while the general task might very well be divided into several smaller tasks, each participant is left free to undergo the task she wants, in an unsupervised way. In other words, in a collaborative setting, the coactors benefit from a community of means to achieve the general task, which implies that all the members of the team has access to the same tools and is involved in the same processes as the others, up to her own will. Moreover, closely articulated to the new technologies and the notion of openness, collaborative teams are supposed to be open, that is, to change over time. The closer concept we have found in English seems to be “Peer production” [2].

This definition, according to which collaboration means unsupervised work, represents the most radical vision of this working paradigm. It is not unrelated to the notion ‘social edition’ proposed by [158] which advocates a complete dissolution of hierarchical structure among DSE teams, so that “every editorial activity, without restriction, might be open to every contributor”, as summed up by Peter Robinson [148]. Robinson then expresses disbelief that such an organisation might enable the team to reach a consensus upon the many decisions that have to be taken regarding “the choice of who the edition should be aimed at; whether it should present an edited text; if so, on what principles should that text be edited; what conventions of spelling, punctuation and presentation should be followed, and many more, right down to decisions on single characters.” And indeed, in practice, the Devonshire Manuscript edition [56] that is made by the tenants of the concept of social editions, is based upon a predefined editorial policy and annotation guidelines that have been defined by a limited number of researchers [1]; also, “following an attempt to vandalize the online edition [published on the Wikibooks platform], the Wikibooks administrators have enforced a review policy on all contributions” [148]. Hence Peter Robinson’s judgement that there is no (and cannot be no) real social edition. And indeed, not all collaborative DSE projects will be as radical as to work on an unsupervised basis, in order to guarantee that the academic standards of accuracy and reliability be satisfied (as it was eventually done for the Devonshire project, i.e. by making sure that the contributions are reviewed by a member of a core, editorial team). Yet, there is another way to consider the ‘failure’, or should we say, the practical limitations the implementation of the concept of social edition had to do

²⁴dictionary.cambridge.org, merriam-webster.com, collinsdictionary.com. Accessed on August 10th, 2017.

²⁵An important research community, whose main topic is the study of activities commonly tagged as collaborative (e.g. Wikipedia contribution, Crowdsourcing), published under the banner of CSCW, for Computer Supported *Cooperative Work*.

with: instead of considering them as evidence that social editions have failed, so far, we might wonder how they could succeed. In particular, the disbelief Peter Robinson expresses about the possibility that a collaborative team might reach a consensus upon the editorial policy is very interesting. Our first question is:

If the hypothesis that the collaborative definition of an editorial policy is impossible, then, in a cooperative DSE teams, provided that the editorial policy will be defined by a small group of editors: how does this group come to a consensus on the editorial policy?

Hence our second question:

Couldn't the editorial policy of a collaborative (in the strong sense of the term) DSE team be not imposed by a small, central editorial board, but instead be the result of a collaborative activity?

Indeed, other experiences prove that unsupervision *does not* imply selfishness or isolation. That collaborative work shall mean that the contributors shall work alone, isolated one from the others, would be paradoxical (and sad, may I say) indeed! Yet it has been established that unsupervised work is indeed prone to self-organised group dynamism, and that collaborative platforms are indeed social places, where behavioural patterns comparable to those in the physical world can be observed. As a first example, the way Wikipedia works is described as a particular case of collaboration, namely, “unsupervised cooperation” [2]. As another example, [189] provides a study of the levers thanks to which organisation arises in contexts where formal organisation is non-existent, and more precisely how the free contributors to an open, collaborative project, can be incentivized to undergo tedious or unrewarding tasks, that are indispensable for the project. Thus if group dynamism can happen in a collaborative setting, why would it not happen when defining the editorial policy of a DSE, or at least, its data structure?

Thus we rephrase the definition of a generic model for the data structure in a DSE setting: the data structure must be adaptable, it must be evolving, and it must be the result of a collaborative process.

Eventually, considering collaborative work in the context of the manufacture of DSE also raises the question of how to handle the situation in which two editors, or more, edit the same part of the corpus in a comparable way, in parallel. This problem, which is part of the general problem of “collaborative authoring” [125], has been solved in the field of software engineering, but is still under investigation for documents, like a DSE, making use of natural language instead of procedural language, where comparison between versions, and subsequent merging, have to be redefined.

2. **Multidisciplinary**²⁶. The term is, at first sight, quite self-explanatory. Still, interesting comments on multidisciplinary can be found in [29]. According to

²⁶We will consider, in this collaborative context, that a *multidisciplinary* team aims at producing an *interdisciplinary* edition, that is, an edition in which the contributions from the different communities are not separated.

the authors, a multidisciplinary team is a team in which the different members come with different research problems, concepts and methods. Also, and very importantly for us, the ‘epistemologic *style*’ of the different members will vary, according to the discipline they come from, leading those members to express information very differently. In a multidisciplinary setting like the ENCCRE project, for instance, it is important to think that a researcher in literature may not aim at the same annotation as a historian, or a historian of science, etc. Moreover, those research problems, concepts, methods, style that set one discipline apart from the others (and the members of that discipline apart from the others), have been integrated by the representants of the discipline in a personal way, which means that among a given discipline, of course, a non negligible diversity of view is to be expected.

The authors add an important remark, that will be at the heart of our upcoming propositions. It is clear that a collaborative team is a social setting with a common centre of interest, upon which diverging views *coexist*; but more importantly, it is a setting in which different views shall be *confronted*. Like in dialectics, it is precisely the confrontation of diverging views that shall drive the activity of a multidisciplinary team, or work: “[C]ommunication between individuals endowed with different conceptual structures is not simply a precondition for attaining interdisciplinary insights, but is an essential component” [29]. From a practical point of view, this means that:

- (a) interpersonal communication shall be a central activity in DSE construction processes;
- (b) generic DSE construction systems shall rely upon adequate communication tools, in order to support multidisciplinary and interpersonal discussions;
- (c) the fact that a collaborative team shall have diverging views, even on something as central and as important as the editorial policy shall not, and can not, be tamed; it is essential to multidisciplinary work, in that it plays a crucial, dialectic role.

3. **Distributed.** The term is opposed to “centralised”: in our context, it means that the editors are not necessarily located at the same place and thus cannot be expected to work synchronously. First, this is another argument in favour of the above conclusions, regarding the necessity to provide the editors with efficient and adequate means of communication. Asynchronism is a problem that is worth considering, from a technical point of view, based on the requirement for the data structure to be evolving. Indeed, as detailed in the paragraph 1.2.2.2, an editor aiming at amending the data structure may want to do so by deleting/adding a new substructure, specializing/generalizing a given element, replacing a substructure by another one. It is quite clear that those operations could be granted to the editor only in the context of single-editor DSE, so that she might make modifications directly on the data structure – and even in this context, this naïve approach of data structure evolution would certainly not be optimal. This approach can simply not be considered in a distributed context in which, due to the asynchronism, editors cannot give real-time feedback to another editor that is amending the data structure...

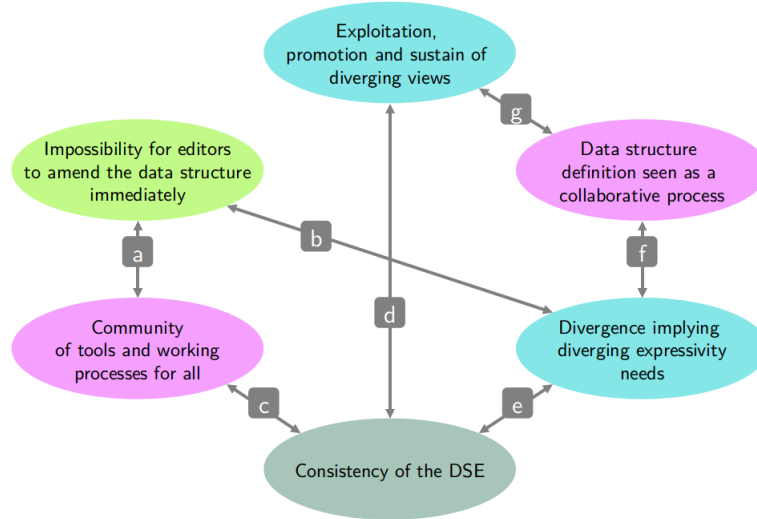


Figure 1.5: Synthetic view of the conclusions of Paragraph 1.2.3.1 highlighting the contradictions between those conclusions.

1.2.3.2 Conclusion: Paradoxical Needs

In the light of the above definitions – and up to their correctness – it appears that a collaborative, distributed and multidisciplinary editorial team has paradoxical needs. Figure 1.5 shows that there is a tension between the individual issues (that are related to the editors taken individually, taking part to the edition of the primary corpus) and the collective issues. The whole diagram may be rephrased as follows: in order to take full benefit from the multidisciplinary setting as defined by Bromme [29], the editors, individually, should be able to tune the data structure as described in Paragraph 1.2.2, in order to be able to annotate freely the corpus according to their personal and disciplinary sensibility. However, the collective work the DSE represents has to exhibit a certain consistency and correspond to a clear editorial policy (contradictions *b*, *d*, *e*, *f* and *g*). Additionally, the manner for the editors to take part to the amendment of the data structure, in order to tune the expressivity of the language of annotation according to their needs, is unclear: it appears that the editors, individually, cannot be given the means to amend the data structure dictatorially, as if they were working alone, and yet the kinds of elementary modifications for the data structure that are interesting for an editor working alone remain equally interesting for one editor working inside of a team... (contradictions *a*, *b* and *c*).

Chapter 2

Common Ground-inspired Digital Scholarly Edition Construction Process

The aim of this PhD work is to propose a generic approach towards the construction of Digital Scholarly Editions (DSE). In the previous chapter, we proposed an exploration of DSE theoretical models, as well as a definition of the most general DSE construction context, which implied, in particular, collaborative team work, multimedia primary corpus, etc. The idea beneath this approach is to uncover some challenges that many DSE projects, even if they only correspond partly to the above model, may face.

The sheer definition of the most general DSE setting lead us to consider some characteristics that are rarely put forward by operating DSE projects, maybe because they are not desired characteristics, or characteristics that, in the current state of available technologies, are seen as causing technical problems – but that still are characteristics of real-life DSE projects.

In particular, while considering, from an abstract point of view, what characteristics the data structure of a DSE project might have in the most general case, we hypothesized that it might not be stable through time, but instead, evolving. It then turned that, even though no DSE project, to the best of our knowledge, exhibits the fact it is based upon a changing data structure, in practice, data structures do change, are refined – not only in the preliminary stages of the construction of the DSE, before any file has been encoded, but in the course of encoding itself. The reasons for that are obvious: it is by trying to instantiate a data structure on the widest corpus that the limitations of that structure are discovered. Moreover, data structures may change not only because they were flawed: since DSE projects often tackle huge primary resources¹, they take years to fulfil. In years, the editorial team may change, bringing new views and new expertise on the corpus, leading to an expansion of the initial editorial policy – and thus, possibly, of the original data structure.

¹The *Encyclopédie* contains no less than 74.000 articles; the team working on Stendhal's papers have edited more than 2000 folios so far, etc.

This generic approach highlighted several needs that may concern a wide amount of projects (the need for making the data structure evolve, and for updating the data consequently, in particular), also highlighted a network of paradoxes in the needs of our hypothetical, generic DSE project team.

Those can be rephrased by the following prescriptions:

1. First, in order to guarantee a certain level of homogeneity and consistency of the edition across a collaborative team, the use of guidelines, setting the fundamental editorial principles for the DSE, but also of a prescriptive data structure *à la* XML schema shall be advocated.
This data structure must be defined adequately with the editor's needs and will, to their field of knowledge, and, in order for them not to have to edit in a way they do not fully approve, to their personal preferences.
2. Second, in the trail of the above, a certain degree of inconsistency shall be allowed between the views, and the annotations, of the different editors, but in a 'controlled way' (giving a tight definition of this vague expression will certainly be one of the outputs of this chapter).
3. Third, the DSE must still be a collective work, that is, more than the collection of the work of its members.

Those prescriptions highlight the tension there is between the requirement of harmony and consistency, at the scale of the DSE, and the individual aspirations and expressivity needs of the editors.

2.1 Current Approaches of the Problem

2.1.0.3 Altruistic vs. Egoistic Data Structures

Actually, we have not encountered any explicit mention of this problematic in the literature on DSE. We can only identify two adverse conceptions of DSE construction, among the existing tools available to the editor, that somehow offer two different practical means to deal with the problem. Those two approaches can be referred to as 'altruistic' versus 'egoistic', following [102].

The **altruistic approach** is exemplified by the Text Encoding Initiative (TEI) [39]. TEI is an XML language based on a huge, cooperatively defined schema, that aims at providing a commonly accepted description of most types of documents or editorial objects. A vast and extremely well documented vocabulary of annotation is thus proposed to the public² by means of a modular schema.

The editors of a TEI-based DSE project then choose among the modules the ones that will be useful for their own project. In the end, TEI works like a more or less universal library of experience-defined annotation structures, out of which each editorial team may generate their own schema.

Noteworthy, apart from the universalising quality of the TEI, being an XML language, it is extensible, in the sense that the TEI user can modify the TEI-based

²TEI is an open format.

schema, either by tuning the existing elements taken from the TEI guidelines, or by adding brand new elements. Yet, this practice, for which the TEI consortium does provide documentation also³, is not encouraged, one of the assets of the TEI being to provide an universal annotation language so that any TEI-based transcriptions be interpretable, exchangeable, computable, without any additional documentation – this aspect of TEI being called *interoperability*. Interoperability is a notion that advocates for consistent annotations not only in the scope of a single DSE project, but across projects. One of the strength of the TEI being to propose an “universal” annotation language, amending that language means loosing the promise of the annotation being universally understandable. Still, in practice, as hinted by [154], tuning the TEI orthodox schema is quite common across the TEI-based DSE projects, and Syd Bauman, coeditor of the P5 Guidelines, insists that instead of aiming at interoperability, a reasonable goal, greatly facilitated by the use of the TEI as a basis for the edition, is interchange, or the capacity to exchange files based on a clear, minimal documentation (inexistent in case of unmodified schemas, well structured else) and interpersonal communication [16]. To a certain extent, this approach is meant as a reasonable balance between homogeneity and singularity, which is reminiscent of the problematic we described earlier. Still, inside one DSE project, a consensus must be reached, in the preliminary stages of the construction of the DSE (before the annotation started), to pick among the modules offered by the TEI optionally tune the selected elements’ definition, and thus define the unique schema of the edition. Schemas are not designed as dynamic also.

The **egoistic approach**, as defined in [102], consists in defining data structures that aim “at expressing as exactly as possible the theoretical assumptions and research interests of one or more scholars”, without any ambition of that structure being used by any other project. A TEI project practising intense tuning of the TEI-derived schema might fall in that category; XML-based projects that rely upon ad-hoc schemas, as exemplified by the Stendhal project associated to this work, even more so. The previously cited Glozz platform may appear as the paragon of this approach. The model of data structure the platform relies upon is proprietary. It is up to the editor to define her own data structure. Actually, several data structures can be instantiated over the same data: the system is thus open to co-existing, diverging and highly specialized structures, possibly defined by different editors.

Still, this model has numerous weak points. First, there is no indication that data structures are designed to be dynamic, that is, that they can be amended. Moreover, the possibility to make several adverse data structures coexist does not imply that Glozz supports the dialogue between diverging points of view: in particular, there is no way to define relationships between the competing structures. In other words, Glozz is a tool that supports plural projects, each pertaining to a particular view on the data, but the paradigm it is based upon does not seem to fit one polyphonic project, where the diversity of views are articulated together.

³See www.tei-c.org/Guidelines/Customization/. Accessed on August 11th, 2017.

2.1.0.4 Another Track to Follow: the Theory of Common Ground

As we can see, tools exist that enable to express a plural annotation of some resources, as the echo of the plural views of the editors. Thus if a multidisciplinary team decided on several data structures that represent the diversity of views of its members (e.g. disciplinary structures), those structures could indeed coexist in the edition, from a technical point of view – independent one from the others, structurally speaking: their instantiation over the same data and the subsequent possibility to publish them jointly would indeed give the illusion of a polyphonic edition.

Yet such a solution would rely on a very poor definition of multidisciplinary collaboration, corresponding to the instantiation by group of a structure that has been defined previously. It does not support data structure evolution either. Moreover, since by definition collaborative teams are prone to change over time, then the new members of the team would be reduced to instantiate a data structure they did not participate to define, while it seems that, because it is so central and conditions the activity of the editors once it is set, the data structure of a DSE project shall be the result of a fully collaborative work.

It is that process of collaborative data structure definition, that requires data structure to be updatable by amendments, that we want to provide the basis for here, so that the editor shall never be reduced to an operating role but shall be, instead, fully author of an expressive and lively edition.

From this point of view, it appears that achieving this goal does not simply demand technical propositions. First of all, it must rely upon a working process that takes into account the way an editorial team may manage its internal contradictions, in order to build a consistent product that does not refrain the expressivity of the editors.

As a starting point, we propose the reader a short study of the theory of *Common Ground* (CG) [46]. Originally, CG is a model of communication, meant to explain how collective tasks can be achieved despite the imperfect mutual understanding among the coactors of the task and the impossibility to reach agreement upon a common representation of the aim of the interactions.

Nota. The question of whether it is possible or not to reach an agreement upon the mental representation of an object is of particular interest in the context of digital scholarly editions. Indeed, representations can be defined as “a formal system [set of symbols] for making explicit certain entities or types of information, together with a specification of how the system does this [i.e. how to formally use those symbols for making explicit that information]”, a way “to capture some aspect of reality by making a description of it using a symbol [so well so that] can be useful” – which is very close indeed of the notion of a *model*. The editorial policy and the data structure, in a DSE setting, precisely work as a *model* of the edition. Can this model be understood the same way by all the contributors to the edition? Will they, after the preliminary discussions that found the DSE work, have the same representation of the project? As established by [66], whatever the way we measure the representations of people involved in the collective realisation of some object, they always differ, which indicates that representations are never shared (in the sense that they are never identical) from one person to the other. Thus: how to conciliate this impossibility to reach an agree-

ment upon a model and the requirement to instantiate collaboratively one common model of the edition?

Clark's CG proposition is in the form of a model of interactions, mainly applied to conversation⁴. CG is based upon the *assumption* by each participant that collective work demands coordination of two kinds: *coordination of content* and *coordination of process*. Coordination of content refers to the fact collective activity demands, from the co-actors, "a vast amount of shared information or *common ground*—that is, mutual knowledge, mutual beliefs, and mutual assumptions". Coordination of process qualifies the synchronization the coactors necessarily have to make to fulfil any action, but also the constant checking that the other still share a mutual understanding of the situation with oneself— that is, updating the common ground. Only this way can sequential actions build one upon the previous others.

This theoretical framework – the distinction between content and process – is not always very clear, or very convincing, particularly when the task to accomplish is communication. It mainly serves to introduce the two core ideas that make the common ground theory promising, from a practical point of view:

1. the common ground, set of mutual understanding, knowledge, beliefs and hypothesis, is updated in the course of the collective activity;
2. collective action is not made possible by the sharing of a common ground, that is, of representations of the work to fulfil, but on the *assumption* of it.

Those two ideas are developed further by Clark, in the shape of two notions: *grounding* and *grounding criterion*.

Grounding. It is the process by means of which the common ground is updated. Depending on the situation and the goal to fulfil, grounding can take different shapes; yet it is supposed be part of any collective action. According to Clark, two factors influence on the shape of the grounding process: the aim of the action and the media that supports its realisation. In any case, the grounding process is based upon *communication*.

Grounding criterion. The theory of common ground does not rely on the hypothesis of a perfect mutual understanding, or the sharing of mental representations, from the coactors: instead, the mechanism that enables the collective action is the fact "the contributor and his or her partner mutually believe that the partners have understood what the contributor meant to a criterion sufficient for current purpose". From there on, Clark suggests that a conversation, for instance, is punctuated by positive or negative evidence of grounding – evidence, that are awaited by those who, at a certain moment, lead the interaction of emit information. Interactions take the shape of an alternation of *presentation* phases, during which one actor emits an idea, suggests a move, etc., and signs of either misunderstanding and refuse (negative evidence of grounding) or agreement and acknowledgement (positive evidence). A sign

⁴So well so that [109] mentions it as a model of conversation, which is reductive since the model is meant to be applicable to multimodal interactions [180]. Truthfully though, the illustrations Clark provides are all based upon physical conversation.

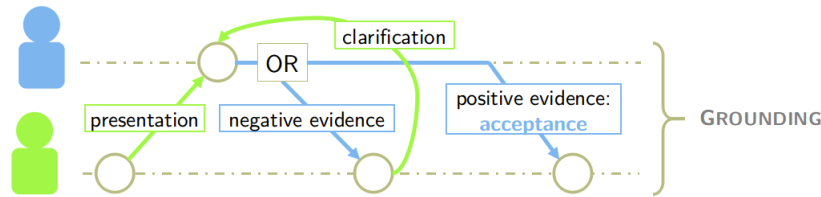


Figure 2.1: The theory of *Common Ground* relies upon two notions: grounding, process by means of which a certain impression of mutual understanding is acquired by the coactors; grounding criterion, referring to the evidence or clues of mutual understanding that punctuate the interaction.

of acknowledgement concludes a succeeding phase of grounding, that can be macroscopically summarized as follows:

$$(presentation) \rightarrow (discussion) \rightarrow (acceptance)$$

For more details, see Figure 2.1. Noteworthy, the pieces of ‘evidence’ of understanding Clark mentions are in fact more clues than evidence (one type of such clues being ‘continued attention’, which can be quite ambiguous) – the ambiguity, or uncertainty of those clues still does not prevent the action from being carried on.

This is a central aspect of the theory of the CG, to be compared to the conclusions of [66]. In that paper, the author studies “shared representations”, spurred by the fact “many are the research studies and the theories in strategic management that rely upon the concept of shared values, beliefs, knowledge or more generally shared representations⁵. In particular, many such studies tend to assert that “the condition for an organization to be operating, its members must converge towards shared representations^{6,7}”. Still, it appears that this “convergence towards shared representations” is, in the literature [66], at the very least, quite limp⁸, so well so that she comes to question the polysemous term “to share⁹”, to the point of doubting of its relevance. To that respect, the author mentions two studies that come to similar conclusions as Herbert H. Clark:

1. The first one [182] “calls into question the necessity of sharing representations to work collectively[: it] suffices that the mutual interest of the actors be satisfied

⁵“Nombreuses sont les recherches et les théories en management stratégique qui reposent sur le concept de partage de valeurs, de croyances, de connaissances, ou plus généralement partage de représentations”, [66].

⁶“[P]our [qu’une] organisation puisse fonctionner, ses membres doivent converger vers des représentations communes”, *ibid.*

⁷Analogically, we have insisted on the importance (or not) of share representations in the frame of a DSE project (see Nota p. 34): shall the editors in charge of editing a document converge towards a shared representation of the document, of the text, and of the resulting digital annotated resources and, to do so, make use of the same language of annotation, for the collaborative work to achieve?

⁸E.g., in [83], it is defined as “thinking, at least up to a certain degree, in a similar manner”.

⁹Thefreedictionary defines to share both as “to divide/to apportion” and “to hold or have jointly. Accessed August 11th, 2017.

so that these actors shall be able to mutually predict their behaviour for the collective action to be possible". A key element in this vision of collective work is the necessity for the participants to be able to predict the behaviour of their partners, which Clark's coordination of process implies.

2. The second [63] proposes the notion of "equifinal representations"¹⁰ to refer to individual, discordant representations that "imply the same actions"¹¹. Once again, this resonates with the fact that, in the CG theory, grounding is followed by action, that is, coordinated action based on diverging views.

We insist here on the similarities between the CG theory and the studies cited by [66], that explicitly state that collective work is not incompatible with diverging representations of the task to be operated. This precision seems necessary, given CG is frequently interpreted as a theory of converging representations (see Paragraph 2.2.1, [118] and [53]).

Bromme [29], already cited for his insights into multidisciplinary work, offers a, enlightening analysis of how diverging views, not only coexist, but can be managed, practically speaking, in a collective work.

[29] is sometimes sourced as providing a redefinition, complementary or alternative, according to the adopted point of view, of the CG theory – not from a linguistic (as in [46]) but cognitive point of view. Compared with Clark who defines the core concepts of the CG theory abstractly, as means to describe any interaction, before illustrating the theory in the context of casual physical dialogues (here casual means: implying no previous knowledge of the discussed topic), Bromme starts by considering a concrete, restricted situation, namely, multidisciplinary scholarly work. As indicated previously, multidisciplinary implies the coexistence, between the members of the coworking team, of different¹² personal and disciplinary perspectives¹³. In this context, very different from Clark's casual conversations, actors do possess crucial knowledge about the discussed object/task. Bromme does not ground his reflection upon the general question "how can partners with imperfect mutual understanding perform consistent interaction?": instead, he decides "to make the difference between disciplinary (and subdisciplinary) conceptual structures the principal point of departure" in his thinking. Indeed, the interactions he focusses on are no longer the place for a sheer, pragmatic agreements, for the sake of the task the partners are involved in, but "processes of confrontation between different structures of knowledge, or perspectives". If Bromme does investigate how coactors, coming from various disciplines may try to do some kind of grounding in the sense of Clark, e.g. by defining a share terminology (that is, a terminology used by all the members of the team, even if the constituting items from that terminology shine differently to the eyes of the individual members), he insists on the fact that multidisciplinary work never, in grounding, reaches a standstill, for

¹⁰"Représentations équitinales".

¹¹"[d]ont les implications sont les mêmes en termes d'action".

¹²Complementary, as is often highlighted to promote multidisciplinary, but also competing, adverse – which, in Bromme's perspective, is not less valuable. See below.

¹³The notion is defined by Bromme as special kind of disciplinary 'knowledge': "'Knowledge' in this context does not only comprise special methods or concepts, but also the epistemic style typical for a discipline or a domain of research activities. [...] This] kind of knowledge will be called *perspective*". Ibid. p. 119.

“a successful agreement on a common terminology in interdisciplinary communication does not dissolve the difference in the disciplinary perspectives”. Even more so, finding such a standstill is not something that shall even be wished: it is the dialectic interaction between individuals possessing diverging (and not only complementary) representations that makes multidisciplinary work valuable, and may let new perspectives arise. In other words, agreement (in the sense of compromise, consensus, since we know from [66] that it is impossible to align representations) is not a reasonable goal: the driving force of interactions is indeed the divergence of views of the actors, associated with active communication and grounding – so one may conclude that divergence of views, and the expression of diverging views, must be *encouraged*.

The vision of *constant inconstancy* may look chaotic, but still, it taints the theory of common ground with what seems to us intriguing and promising colors. The ground of Common Ground is not a one in the geological sense, some solid basis that builds by the accretion of all the previous agreements, working, from there on, as a shared gold truth. It is a fugitive ground, that has limited lifespan, and that is in permanent renegotiation. The *common ground* is not a pyramidal, but a transitory construction.

2.2 Operating the Common Ground Theory

In the previous paragraph, it appears that the editing activity could not be considered from a technical point of view, for instance by focussing solely on encoding formats and editing interfaces: this would obliterate an important, but rarely considered challenge, that is, how to make sure that all the editors, at each step of the DSE construction, be its builders (architects and craftsmen), and not be limited to instantiate a data structure that, in some cases, they did not even define. Given the generic context we consider for DSE construction, by a collaborative, distributed, multidisciplinary team, interaction between the editors had to be considered. In particular, it appeared that to be up to the challenge cited above, even if the goal of the editing team is to produce a consistent edition, the diversity of views on the way this edition should be shall be expressed, encouraged and concretely valued. This paradox can be solved by considering how the two scales, individual and collective, articulate one with the other. The Common Ground theory provides the following insights into this articulation:

1. [46]: a team exhibiting diverging views upon the edition can work towards building the editorial policy, despite those divergences: uniformity of mental representations is not necessary, so long as the editors agree on a transitive, consensual policy resulting from the grounding, that may not cover the ‘sum’ of the diverging views on the policy in detail...
2. [29]: ... as long as those diverging views are still expressible and can be confronted during the grounding, seen as a perpetual process.

Before translating those two points in more practical terms for a DSE project, we investigate how the theory of Common Ground has been operated in Computer Science.

2.2.1 The Common Ground Theory: Applications

Since the early 1990's, the Common Ground theory was adopted (and adapted) in Computer Science research, mainly in three research communities: *Computer Supported Cooperative Work* (CSCW), *Computer Supported Collaborative Learning* (CSCL) et *Computer-Human Interaction* (CHI). Those three communities share some characteristics, like being human-centered and investigating the use of digital tools by individuals and teams¹⁴. They seek behaviour models and tools to assess the quality of the interaction between an user, or a community of users, and the machine, in a variety of tasks that often deal with 'knowledge creation' [62, 124, 87]. In the corresponding literature, Common Ground is, to the best of our knowledge, considered from three points of view:

1. First, the Common Ground is considered as a first-order research object. Its underlying premises are questioned and deepened. For instance, [118] tests the assertion that grounding is hampered when the coactors are distributed¹⁵; [22] questions the general notion of "communication media richness" that Clark indirectly defines¹⁶. In the same tone, [179] studies the impact of video transmission on grounding.

[109] is an example of a general call into question of the CG theory, both as a model of communication and as a tool for computer scientists – we will detail that below.

Other sources operate methods that aim at getting a better understanding of the grounding phenomenon: for example, [54] experiments on the influence of repetition over the "growth of common ground", measured by the combination of the quantity of information memorized during the interaction and the qualitative feeling to have shared information.

2. Second: the CG theory is adopted as a means to decrypt interactions. Two non exclusive variants can be isolated:

- the CG theory is used as a means to analyse the nature and the sequentiality of the interactions that are enabled by a given media. [114] proposes a typology based on the notion of grounding for feedback interactions in a learning interface. [131] employs CG in order to explain the content and the aim of interactions that are commonly classified as "off-topic" – and are therefore little studied – in online learning interfaces;

- the CG theory is operated as a means to evaluate interactions in a communication interface. Convertino *et al.* [52] define a metric for common ground,

¹⁴Group cognition, which the Common Ground theory is part of – especially as defined by Bromme [29] –, because it enlightens both the needs and practice of communities of users, have raised particular interest [165].

¹⁵Indeed, one aspect of the study of [46] concerns the correlation of grounding with the media used for the interaction. As a result, Clarks provides a list of eight "constraints", so that the lack of one of those constraints demands the developments of particular grounding techniques, whose cost can be estimated (hence the notion of *cost of grounding* proposed by Clark. Physical presence is one of them. See *ibid.* p. 142.)

¹⁶It can be defined by the list of constraints verified by those media – see note 15.

and then [53] make use of that metric to evaluate a group decision-assisting interface.

3. Third, the CG theory is called for to predict the interactions that may lead to the fulfilment of a communication-based task, or to limit those interactions in order to enhance their efficiency, in terms of grounding. One example of that can be found in [104], who makes the assumption it is possible to force interactions to follow the schema represented on Figure 2.1 page 36: the users are asked to declare, explicitly, whether they have (or think they have) understood a given proposition that was *presented* in the course of the interaction, then to give their position towards that proposition, before any interaction can go any further.

Those diverse interpretations and operations of the CG theory do raise some criticism. In particular, the last use of the CG as a predictive and coercive tool is orthogonal with Clark's views, who explicitly states that the perspective he defends is incompatible with predictive models of dialogue, according to the review by M. A. Walker [180] of his work, *Using Language* [45]. Yet, according to Walker, Clark's knowledge of dialogue planning reflects the state-of-the-art of 1971; significant progress having been made since, Clark's opposition to such interpretations of his theories shall be nuanced.

Stronger criticism of Clark's theory itself, and of the use of that theory in the field of *CSCW* can be found in [109]. His conclusions are the following:

1. About the CG theory: as a conversation model, it does not provide a description of situations where comprehension is hard (multiparty interactions, with brief and overlapping participations).
2. About the application of the CG theory in *CSCW* literature: Koschmann [109] warns that "Serious problems arise when one begins to treat common ground as if it were a singularity, a possession of the participants, a place, an arrived at state, in short, a noun instead of a verb. [...] It is not a thing that can be measured, either directly or indirectly". Hence the question whether it is relevant to try to measure the common ground. Instead, the common ground can be understood and used as a theoretical tool that can guide the conception of interfaces (for instance) – and the evaluation of those interfaces shall then not consist in an evaluation of how the concepts that come from the CG theory translate into the interface, but by means of traditional usability evaluation methods.

Let us note that Koschmann's critical paper is not beyond criticism either. First, the author focusses on Clark's publications ([47] and [46]) to question the theory of Common Ground – which indeed are the most cited in the *CSCW* literature – while at the time the articles of Bromme were also cited (e.g. [92]). And Bromme provides enriching clarifications to Clark's theory. To illustrate this point, let us take an example. In particular, Koschmann mentions, as a weak point of the CG theory, that it does not apply to interactions in the course of which the environment changes, for "[t]his changing environment is at odds with Clark's contribution theory in that the theory would seem to require that contributions to common ground aggregate over time and remain relevant". Indeed, one can find, in the introduction of [46], the notions of

“update” and “accumulation”, which can be interpreted in the direction proposed by Koschmann; still, one can also consider that, Clark’s theory precisely aiming at enlightening interaction in which the enunciation is ambiguous (and therefore necessitating, to be understood, to refer to the context of the enunciation and to the “accumulated” common ground), a proposition cannot be detached from its context. Thus, by depossessing propositions from their context (so as to make them permanently relevant), and by turning the common ground into a container of such context-free propositions, Koschmann simplifies the CG theory to the extreme and goes against Clark’s views. The problem is evacuated when one considers Brommes’s reformulation of the CG, since it is thus clearly exposed as an abstract, transitory structure, endlessly questioned and redefined, at the crossroads between diverging points of view.

Similarly, Koschmann intends to fault the CG theory by demonstrating that, at the end of a complex interaction requiring learning by practice (in the context of surgery), the actors have not memorized properly what should have been learned. Yet Clark’s theory is not a learning, but an interaction theory; moreover, the fact that the complex interaction (surgery training) should have happened even though the comprehension of the task by the different coactors is obviously imperfect precisely *speaks in favour* of a theory that institutes the impression of understanding, to a degree sufficient for the interaction, as the driving force of the interaction.

We can even go as far as saying that, by founding his argumentation on an incomplete definition of the common ground, Koschmann is less critical about the theory itself than about a personal perception of that theory. Actually, the CG theory often suffers from excessive simplification, or even contradictions. The source of this phenomenon can be found in the articles of Clark themselves: in most publications about the CG, Clark starts by providing a simplistic definition of the concept as “[Mutual] knowledge, mutual beliefs, mutual assumptions”, following [166] as quoted in [100], and then makes his proposition more dense all along the article, by introducing corollary concepts (grounding, grounding criterion, etc.). Yet that simplistic, initial definition is the one that can generally be found in the computer science literature that make use of the CG, more or less adapted¹⁷ – which tends the CG to be generally regarded as theorizing the convergence of adverse representations.

Still, let us mention some articles that propose a more nuanced or critical definitions of the CG: [52] proposes an alternative of the common ground, meant to translate a conversation model fit for Computer Mediated Communication into a tool usable in CSCW; [104] elaborates a similar definition out of both Clark’s and Brommes’s contributions, as “representations [...] similar enough to be considered variants of the same representation. [...] People communicate on the basis of imperfect assumptions about the others’ perspectives”. Yet the application of such definitions is not simple and their asperities are often rounded off by this exercise. For instance, the last paper cited above subsequently proposes a functional definition of the common ground: an interface compatible with grounding shall support “construction of shared problem representation” [104].

¹⁷Thus the following definitions can be found, most often without any attempt at detailing it any further: “mutual knowledge, beliefs, and assumptions of the participants in a conversation” [118]; “mutual understandings and beliefs that arise from similar background and experiences” [179]; “state of mutual understanding” [22]; “mutual knowledge, beliefs, and protocols that partners share”, [54]...

2.2.2 CG-inspired collaborative data structuring in a DSE setting

In the light of the above state of the art on the CG theory and its applications, we hereafter propose a DSE construction process inspired by the philosophy and the conclusions of the CG. We give a short summary of the process, before dwelling into more details.

2.2.2.1 A Double Interpretation of the Common Ground

Our approach is mainly user-centered.

Data structuring for DSE consists in the definition of a data structure and in its instantiation on the digital corpus. Based on the history of the four DSE projects associated to this PhD work, data structures may change, or more precisely may evolve during the course of the DSE construction; we have also proposed several basic operations by which one may want to amend a data structure, regardless of the working organisation chosen for the project. By “expressivity needs”, we refer to the motives for making the data structure evolve. Indeed, in our view, the reason why an editor shall want, at time $t + 1$, to make the data structure evolve, is either in order to be able to express information that she cannot express at time t , or to refine the way information can be added onto the corpus as defined by the data structure at time t . Then, by considering the editor team as a group, oriented towards the realisation of a collective task, and the editor as an individual involved in that team, we came to the following question: given the editors, in particular in a multidisciplinary setting, will perceive the edition project differently, and thus have diverging expressivity needs:

Can the editors, involved in a collective task of data structuring, be given, individually, the possibility to amend the data structure accordingly with their personal expressivity needs, while guaranteeing a certain level of consistency for the project as a whole?

To answer that question, we take inspiration from the CG theory as defined earlier, based on the contributions of Clark [46] and Bromme [29].

As an interaction theory, CG partly answers the above question. Clark’s theory explains that sharing representations is not necessary for the fulfilment of the collective task: imperfect agreement, relying on clues given along the interaction, suffices. In our particular context, we face a very concrete instance of representation: the data structure itself, which is a model of the edition in McCarthy’s double sense (model for/model of) [119]. The CG may thus indicate that the perfect agreement, understanding, and mastery of the whole language of annotation, defined by the data structure, by all the editors, may not be considered necessary – so long as they give clues that they understand and master it *enough*, i.e. to a degree sufficient for their contribution to the edition. The above may be interpreted two ways: **(a)** the data structure could be an aggregation of specialized substructures, mastered only by a few editors – those who are at ease with them use them, those who are not, do not – in that case, the data structure will match no individual representation and needs by excess (but the specialized parts of the structure are meant to match some individuals’

representation and needs); **(b)** the data structure could be limited to a minimal vocabulary and grammar covering only general concepts, upon which the editorial team reached a consensus, e.g. for the needs of a first and superficial annotation of the corpus, and for which all the editors have shown a feeling of understanding and mastery (they are all above to instantiate that structure) – in that case, the data structure will match individual representation and needs by default, but will support a consistent, collective work.

The point (a) above is to be related to the idea advocated by Bromme [29], that the diversity of representations is irreducible in multidisciplinary settings – of which it is, even, con-substantial. Moreover, Bromme suggests that the “emergence of new perspectives” cannot be obtained but by the permanent confrontation of diverging views: for instance, the coexistence of specialized substructures as sketched in (a) shall not be understood as independent, parallel structures, whose divergence is highlighted, discussed, possibly leading to either their rejection (if their existence did not convince) or general adoption (as described in (b)).

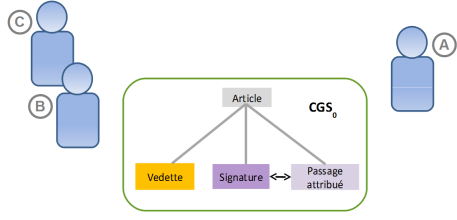
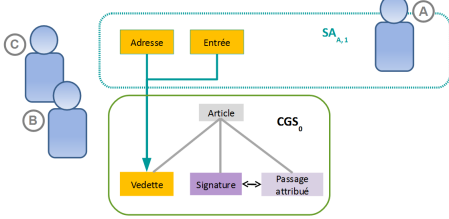
From there on, we can imagine a data structuring process **(c)** in which the editors, individually, shall be given the means to define “personal”, experimental data structures fit to their needs by amending the existing data structure; in which they would be able to instantiate, in a sandbox fashion, those amended data structures; to bring them to the discussion with the other editors, with the perspective of the amendments to be widely adopted by a wider editorial group, or even by the team as a whole. This process can be compared to Figure 2.1 page 36.

Metaphorically, we propose to make the process described in (c) rest upon a bipartite data structure: we will refer to the Common Ground Structure (CGS) for the part that corresponds to the point (b), that is, a data structure that, at a certain moment, has been considered acceptable by all the editors, and Structural Amendments (SA), for the structural refinements proposed by the single editors, and whose acceptance by the editorial team shall impact the CGS, which shall then evolve.

In the following, we provide an illustration of how a DSE team shall operate with such a structure. We then investigate the technical problems this approach comes with.

2.2.2.2 CG-inspired Data Structuring: Illustration

For the sake of the illustration, we propose the following, simplistic structuring case: an editorial team made out of three editors (A, B, C) has defined an initial data structure, which assesses that in an *Article*, there are *Vedettes*, *Signatures* and *Attributed Passages*, which can be related to a *Signature*. This structure is, at the starting moment of this scenario, instantiated by all the editors: it is the initial CGS. The scenario is illustrated hereafter.

Instant	Action	Illustration
$t = t_0$	Initially, the three editors A, B and C define a common data structure they instantiate on the corpus. That structure is the initial Common Ground Structure CGS_0 .	
$t = t_1$	The editor A considers relevant to distinguish between the vedettes in capital letters (the “adresses”) and those in small capital letters (the “entrées”). Those two notions are special cases of the notion of <i>Vedette</i> . The editor A thus defines two new data types <i>Entrée</i> and <i>Adresse</i> that both specialise <i>Vedette</i> , as a structural amendment $SA_{A,1}$. This amendment being made by one single editor, it does not belong to the CGS. The editor A can still instantiate the amended structure on a personal, sandbox copy of the corpus.	

<p>$t = t_2$</p>	<p>The editor A decides to submit $SA_{A,1}$ to the other editors. They discuss the types introduced by A, but also methodological aspects (how to identify and distinguish between “addresses” and “entrées”?), retro-compatibility of the amended structure with the data already instantiating CGS_0, based on the examples she can provide. Time is given for further testing.</p>	
<p>$t = t_3$</p>	<p>The editor A further decides to redefine the data structure enabling the attribution of the articles. She claims that a type representing a collaborator is needed. This idea results in reshaping the structural amendment into $SA_{A,2}$, that subsumes, in this particular case, $SA_{A,1}$.</p>	
<p>$t = t_4$</p>	<p>The editor A submits $SA_{A,2}$ to the others.</p>	

$t = t_5$	<p>At the end of the interaction between the editors, the amendment is partially adopted. The type <i>Collaborateur</i> together with the relation with <i>Signature</i> are integrated into the CGS, which becomes CGS_1. $SA_{A,2}$ changes also; in this particular case, it is brought back to its previous state $SA_{A,1}$.</p>	
$t = t_5$	<p>The editor B is not convinced by the distinction between <i>Adresses</i> as proposed by A and <i>Entrées</i>. This amendment does not, at least so far, integrate the CGS. The editor C, on his side, decides to experiment with the type <i>Adresse</i> only. To do so, she creates $SA_{C,1}$ based on $SA_{A,1}$.</p>	

Nota. The temporality along which the CGS and the SA evolve is not the same: SA may evolve without CGS changing. The evolution patterns are not linear, and can go backwards, either because a SA is partly accepted (see $t = t_5$) or because the editors decide so, in particular after having discussed with the other editors.

As a summary, the editing process we propose makes dynamic structuring possible, based on a central, core structure that can be instantiated by all the editors (the CGS), but also enabling the definition, testing, and proposition of alternative structures (relying upon SA), that are debated and shall possibly make the CGS evolve. In this sense, not only the structured data, but also the data structure itself, shall result from a collaborative work – which, to the best of our knowledge, is a novel proposition. This process, however, raises quite a few challenges and stakes, that we describe here.

2.3 Conclusion, Stakes and Challenges

In this chapter, we studied DSE construction from an abstract, and yet experience-grounded, generic point of view. We aimed at considering the most general configurations for DSE construction, in terms of edited corpus, editorial policy, editorial team, etc. This approach was meant to identify general problems DSE teams face, that may not be mentioned very often in the literature. We came to consider two such problems:

1. DSE projects rely, most of the time, upon predefined data structures, or schemas. Schemas condition the information the edition will eventually contain. As such, it seems quite important that all the editors of a given project should have participated in the definition of the data structure, since it sets the boundaries of the information they will then be able to capture in the structured data. Yet, even in the context of social editing experiments, data structures are not collaboratively defined: they are set as a preliminary step, often by a limited number of editors, who subsequently write guidelines for the other editors that annotate the digital representation of the primary accordingly, in a second step.
2. The above-mentioned two-steps process raises another question. Based on the history of four DSE projects associated with this work, we identified that data structures, far from being intangible, once-and-for-all objects, need to evolve during the history of the DSE. Indeed, data structures are models of the edition, of the primary corpus, and models often change when they are confronted with the reality they represent in the long term. And yet, data structure formalisms do not come with any assistance for data structure upgrade which is then problematic in the context of DSE in particular, since upgrading the data structure implies, quite often, that the structured data representing the edition are outdated and needs appropriate update also. When data structures are effectively updated, in DSE projects, and adapted to the better knowledge editors have got of the corpus, the structured data update is done either manually or thanks to *ad-hoc* scripts, if relevant, which demands either humongous amount of time or the availability of technical staff members for writing those scripts. In case none is at hand, the project has to go with an misfit data structure – which seems to be a quite common situation.

The challenge resulting from those two elements was then to define a means to enable data structure definition, and evolution, in a collaborative setting. It appeared that a data structure formalism up to this challenge should be able to reflect a consistent editorial policy, and at the same time meet the expressivity needs of the editors as individuals.

To solve this paradox, we developed a new interpretation of the concept of Common Ground. In our context, a data structure can be regarded as a representation of the edition to be made. Literature on the Common Ground indicates that no unique representation of the edition will arise; on the contrary, new perspectives may develop from the confrontation of diverse representations. However, editors may agree on an ephemeral feeling of mutual understanding, based on the use of a basic, common annotation language, or upon the confidence that one of them can lead an expert editorial project, in the frame of the common project.

We can rephrase this more concretely. An editorial data structure can be composite. It can be made of an evolutionary core structure and temporary peripheral structures. The core structure is made of types and links upon which the whole team of editors agreed at an instant t . This agreement could be based upon the fact that they share the impression that they are able to implement it. Peripheral structures are proposed by any editor, and are defined as modifications of the core structure.

Such peripheral structures are not meant to coexist independently. A typical scenario follows.

1. An editor instantiates S , which is the core structure.
2. While annotating, she notices that one of the types in S is not adequate for the content to be annotated. She transforms S into a peripheral structure S' , in which she defines a new pattern of types in place of the former one.
3. She argues in favour of S' before the other editors, through the edition tool, by showing use-cases and instance samples – the other editors reply.
4. S' is either accepted or rejected by the community of the editors. In case of acceptance, S' becomes the new core structure.

This scenario raises technical and practical challenges. Obviously, such an intense communication-based process demands efficient means of communication, even more so given the object that will be discussed will be complex: data structures, editorial policy, annotation patterns. Also, since schemas are highly formal objects, it may be relevant to assist their definition by means of user-friendly tools, that alleviate the difficulty and make the definition intuitive. Those are important aspects, that shall require serious research in interfacing, will not be treated in this work.

Instead, we will focus on more upstream challenges, that condition the feasibility of the whole process. Those challenges are twofold.

First, since one of the purposes of the above process based on CG is to enable the evolution of the data structure, thus, it seems relevant to go for highly expressive data and schema models, or even multistructured data models. Indeed, it is tautological to say that if evolving data structures are relevant, it is due to the fact one cannot pre-empt the data structure that will be needed in the future. Thus, to make it simple, proposing a mechanism to make data structures evolve for XML, for instance, which is more or less limited to the expression of *hierarchical*, single layer annotation, may not be relevant: who knows if the hierarchy will always be an appropriate shape in the future of a DSE project? It has been hinted that in a generic approach, we could not obliterate projects in which several annotation paradigms were to be instantiated at once, on the same data – which is another argument in favour of highly expressive data models, like the ones that belong to the multistructured data family [141].

Second, and more importantly, as mentioned before, changing the data structure of a DSE project will, most often, lead to data conflicts. Consider the situation in which there is a schema S and I_S the set of documents instantiating that structure. If the data structure is amended so that the new data structure S' does not subsume¹⁸ S ,

¹⁸Schema subsumption being defined, in general, by the inclusion of the instances of the subsumed schema in the set of instances of the subsuming schema.

then, possibly, I_S will not be validated by the new schema. In other words, providing tools for amending schemas cannot go without providing tools for updating the instances.

In our case, as indicated above, we do not want the schema to evolve, step by step, in a linear way. What makes the DSE constructing process introduced in Paragraph 2.2.2 adapted to collaborative work is the fact schema evolution is not centralised, but initiated upon the initiative of the individual editors, by the definition of alternative structures (amended versions of the CGS), and then evaluated by the other editors. Thus, in our proposition, the amended structure does not replace the other, but, in the beginning, coexists with it, and only after having been discussed will it be decided whether the amendment is accepted or not. If the amended structure is accepted, then we must be able to translate the data that was conform to the old CGS into data compatible with the new CGS, as much as possible. Hence the need to be able to derive, from the transformation of a schema into the next one, a transformation of the old structured data into a shape compatible with the new structure.

Importantly, not only do the structures coexist during the interaction, but also instances of the competing structures as well: the editor who proposed the amendment may instantiate her own structure, be it to undergo a personal investigation on the corpus or to illustrate the validity of her proposition. If her data structure is to be abandoned, it would be great if her data could be translated back into a shape compatible with the CGS, so that all of her work shall not be lost.

In other terms, from a situation in which there is a CGS schema S and its instances I_S , it means that a bidirectional transformation Ff/Bf shall be derived from the schema transformation SA defined by an editor, so that the image of I_S by Bf is structured data $I_{S'}$, validated by S' , and so that Bf/Ff shall work as a synchronization between I_S and $I_{S'}$ (any information added on one side is translated, if possible, on the other side):

$$\begin{array}{ccc} S & \xrightarrow{g} & S' \\ \hookrightarrow & & \hookrightarrow \\ I_S & \xleftrightarrow{Ff/Bf?} & I_{S'} \end{array} \quad \text{and } (g.S = S') \hookrightarrow g.I_S.$$

Meeting those challenges would open promising perspectives. Editors would be given ways to fine-tune the existing core structure, or to propose new peripheral structures to enrich the initial editorial project and to experiment on those structures. More fundamentally, if we had ways to translate structured data from one structure to another by the means of a bidirectional transformation, then even if the editors were working on peripheral projects, data from those side projects would be converted into a shape compatible with the core structure; thus the collective edition, validated by the core structure, would keep progressing. Eventually, if a peripheral structure was accepted and the core structure updated, editors would be given the possibility to update the *data* instantiating the obsolete core structure; otherwise, the work done by the proposing editor would still be preserved, by being translated into another shape, respectful of the collective editorial policy.

As a conclusive remark, it must also be mentioned here that this PhD work, that is focussed on the collaborative definition of the *data structure* underlying a DSE, does

not frontally consider the yet crucial problem of “collaborative authoring” [125], that is, the technical problem concerned with how several editors may instantiate, in parallel, the same data structure, and how to make it possible to end up with one consistent annotation. Indeed, collaboration authoring, that has been well studied in the field of software engineering, raises many additional difficulties in the domain of literary studies and natural language documents [125]. While software engineers have developed very strong methodology and tools for permitting team-based software development, enabling developers to “share documents, divide them into complicated subdocuments, edit them in parallel, merge editing changes semi-automatically, and recombine the subdocuments into a cohesive and correct whole”, tools enabling to track change in a structured document making use of natural language and in the end somehow ‘merge’ those change are being investigated (e.g. [133, 176]). The collaborative authoring problem, indeed, is made difficult by many dimensions, one being the fact that the “correctness” of any merging of natural language information sources is a tough notion to define, and even more so, to check.

This problem is, of course, one of the problems that shall have to be taken into account for the implementation of any collaborative edition platform, in practice. It shall even be taken into particular account in a situation where two editors may not only instantiate the same data structure, in parallel, on the same passage of the edited work, but may even annotate the same passage according to different structures that shall translate one into the other: then, on each side, shall the manual annotation done by an editor be compared, or merged, with the annotation that has been semi-automatically translated from the manual annotation done on the other side, and conversely...

In the frame of this PhD work, we have chosen to leave this problem apart, by considering a very simple temporal model for user interaction, that prevents from generating such conflicts (see Paragraph 10.2 page 254). In the following, we will consider that only one editor can be active at a time. As one can guess, this simplifies greatly the collaborative authoring problem. Yet, it leaves many other problems open:

1. First, we will consider the problem of data expressivity. In a setting in which we consider that the editorial team needs to be able to shift from a data structure to another, it is natural to aim at providing editors with the means to express any kind of annotation – which means, to provide them with very expressive annotation models – and to be able to validate it. Part II gives a detailed presentation of the annotation model we propose: extended Annotation Graphs (eAG) and Schemas (SeAG). Extended Annotation Graphs is a stand-off markup model, which means that an annotation takes the shape of a graph (made out of nodes and edges). While, compared to classic inline (tag-based) annotation models, graphs are not easy to express (a dedicated interface shall be required to draw an annotation graph directly, for instance), this graph formalism offers good calculation properties and is compatible with a novel validation mechanism, that is based upon the notion of simulation [38], and that permits validation to be checked very efficiently, even for very dense and complex annotations.
2. Second, we will focus on how to express extended Annotation Graphs easily, without needing a dedicated interface for that purpose. Part III introduces Linear extended Annotation Graphs (LeAG), that is a classic, inline markup syntax,

that enables to define eAG annotations on textual resources. We also define an efficient parser for translating a LeAG into an eAG, and hence, benefiting from SeAG validation.

3. Eventually, in Part IV, we will define how schemas shall be amended, and how the amendment of a schema can be interpreted as a bidirectional transformation between the instances of the original and the modified schema.

Part II

A Schema-aware, Multistructured Data Model Tuned for Scholarly Annotation: the eAG/SeAG Model

Chapter 3

Introduction

3.1 Preliminary: Notation

This work makes use of the notion of directed, labelled graph. The most useful notations for handling such graphs are given below.

Constitutive items. A directed, labelled graph is a tuple $G = (V, E)$, where V is the set of nodes and E the set of edges of G . Be \mathcal{L} a set of labels. Then there is a function $label : E \rightarrow l \in \mathcal{L}$ thanks to which a label from \mathcal{L} is associated to the edges of G .

Definition 3.1.1: Summit, end and node degrees. Be $G = (V, E)$ an oriented labelled graph. Be $e \in E$. The edge is oriented, which means that it connects a unique node called its **summit** to a unique node called its **end**, respectively denoted $sut(e)$ and $end(e)$.

Be $v \in V$. The **in-degree** of a v , denoted $in(v)$, is the number of edges it is the end of. The **out-degree** of a v , denoted $out(v)$, is the number of edges it is the summit of. The **degree** of a v , denoted $deg(v)$, is the sum of the in- and out- degrees.

Definition 3.1.2: Roots and leaves of a graph. Be $G = (V, E)$ an oriented labelled graph. A node $v \in V$ is a **root** of G iff $in(v) = 0$. A node $v \in V$ is a **leaf** of G iff $out(v) = 0$.

The set of the roots and leaves of a graph G will be denoted $root(G)$ and $leaf(G)$ respectively hereinafter.

Definition 3.1.3: Rooted and single-leafed graphs. Be G an oriented labelled graph. G is **rooted** iff $root(G)$ is a singleton. G is **single-leafed** iff $leaf(G)$ is a singleton.

Notation: Linear graph. The notation $v_1 |e_1 \dots| v_N$ will be used to denote a linear graph made out of the nodes $\{v_i\}_{i \in [1;N]}$ and the edges $\{v_i\}_{i \in [1;N-1]}$, $N \geq 1$, so that for all i , $v_i = sut(e_i)$ and $v_{i+1} = end(e_i)$.

Notation: Subgraph of a graph. Be a graph $G = (V, E)$. The graph $G_{in} = (V_{in}, E_{in})$ is a subgraph of G iff $V_{in} \subseteq V \wedge E_{in} \subseteq E$. The fact G_{in} is a subgraph of G will be denoted $G_{in} \subseteq G$.

The expression $G_{in} \subset_{v_2}^{v_1} G$ means that the graph G_{in} is rooted, single-leafed, is included in G , and possesses $v_1 \in V$ as a root and $v_2 \in V$ as a leaf.

Let us now come to the heart of our purpose, and introduce step by step the eAG data model. We start by a short description of Annotation graphs, which eAG is based upon, then propose some amendments to this original model, and then give two examples of the expressive power of eAGs.

3.2 Outline of this Part and Main Contributions

As we have seen in the Introductory part of this document, there is a need for an annotation format, language, or model, that shall be schema-aware, and that shall support data structure evolution at the same time, in order to make it possible for long-term DSE projects to update the structure of the critical annotations in the course of their manufacture, either because it, after further consideration and operation, appears not to provide an adequate description of the primary corpus, or because the editorial policy of the DSE project has changed.

When designing an annotation, or data, model meeting those expectancies, one may consider that this data model must as versatile as possible. Indeed, it would not make much sense to acknowledge that the structure of data resulting from scholarly annotation is subject to unpredictable change and to limit the structure of the data to a restricted kind of annotation – for example, to hierarchical annotation. Hence the following panorama of the most versatile data models for annotation at hand, that all go beyond the hierarchies, gathered under the umbrella term ‘multistructured data’ – as possible sources of inspiration for the desired, evolution-friendly data model.

Multistructured (M-S) data models have been a hot topic for over a decade. Correlated to the rise of Digital Humanities, they ground on the fact that a single hierarchy is not always sufficient to represent annotated resources [39], contrasting with the setting of XML-based languages as a standard for scholarly annotations. Hence, “multistructured” is then to be understood by comparison with XML: annotating somehow means *structuring* data (a well-formed XML document fits into a tree *structure*); “multi” suggests M-S data models handle multiple, interlaced hierarchical annotations over the same data. Many models have been proposed [141]. However, the enhanced expressiveness resulting from less constrained structural foundations, compared to XML, comes at a cost: M-S models often lack a corresponding data model.

Indeed, validating highly expressive data is challenging, due to a general trade-off between data models expressiveness and algorithmic complexity. The NEXPTIME

complexity of OWL/DL inference [146] will serve as a striking example of how costly the validation of highly expressive graph-structured documents can be. This trade-off is so pregnant and restrictive that it applies to XML [170].

After the description of the state-of-the-art regarding data models for annotation, we introduce our first main contribution, that was presented at the DocEng conference in 2016 [15], namely, extended Annotation graphs (eAG). eAG is a multistructured data model, tailor-fit for scholarly annotation. In particular, it goes beyond the classic, technical problems of overlap and expression of multiple, independent layers of annotation; it also provides the editor with tools to reify distant relations and to make accurate quotes within the data, both useful for the study of intertextuality and the identification of web-organised information. It also provides the editor with means to insert critical notes and comments within the data, to interlace this critical, textual data with the primary data, while making sure that the two kinds of data, primary vs. critical, shall not be confused. In the end, eAG comes as one of the most expressive data models available.

We also introduce our second contribution, that is, a schema language for eAGs, called SeAG, based on a novel validation mechanism that bypasses the traditional trade-off between expressivity and complexity. Indeed, we have identified the *simulation* relation, first used for the structural description of semistructured data [37], as a promising mechanism for eAG validation. Then, based on a coupled representation for SeAGs and eAGs, we show that given the representation of a schema, only valid eAGs can be represented: this is “validation by construction”. This enables to guarantee the validity of rich M-S data *without algorithmic check*, bypassing the trade-off between expressiveness and complexity, when schema definition can precede annotation. Additionally, the eAG/SeAG model is compatible with classical, *a posteriori* validation, in which case the schema may be given after the eAG was built, and validation checked afterwards. In this case, checking whether an eAG is valid against any schema can be decided in polynomial time ($O(|edges| \cdot |nodes|)$). We finally proved that for hierarchical data, SeAG syntactic validation is not less straitening than Relax-NG.

At the end of this part, the definition of a very expressive, schema-aware data model for which the time-complexity of the validation is not problematic will have been defined – which constitutes a first step towards a data model supporting data structure evolution, that will be completed in the next parts.

Chapter 4

Related Work

4.1 Multistructured Data and Validation

Currently, most scholarly editorial projects adopt the XML-TEI markup language for annotation. Such a choice has solid foundations: the TEI project provides researchers with rich, collaboratively designed and documented schema modules dedicated to almost all sorts of heritage annotation, along with a tool to cherry-pick such modules, tweak (to a certain extent) and assemble them into a custom schema¹. As for annotation itself, from a practical point of view, (free) user-friendly XML edition tools abound on every hand. Due to its ascendancy that is rooted in printing formats [151], XML is a natural candidate for textual description. Models for text encoding need to fulfil a few prerequisites: support for linear characterization (along *the* textual dimension – if unique); representation of inclusion relations (e.g. to describe *the* material structure of a corpus – if unique), which XML does well (in case of uniqueness of both preceding attributes of the corpus): due to the inline nature of XML, not only are the elements ordered along with the text, but also are they maintained in their context; inclusion has been decided to be represented by nesting.

Still, in practice, tree-based formalisms are known not to fit some advanced but common textual description patterns [151, 145, 39]. In particular, XML suffers from: the inability to manage overlapping elements; a weak, non syntactical representation of links, that need to be validated separately²; the expression of inclusion by nesting, that raises the question of how to express accidental, non inclusive nesting, very common when annotating according to more than one paradigm simultaneously. Several propositions have been made to conform TEI-XML with more expressive data models [39, 33, 42]; although interesting, those propositions either fail to tackle part of XML inherent weak points³ or are not straightforwardly compliant with the many useful validation, querying, transformation tools from the XML galaxy [141].

¹<http://www.tei-c.org/Roma/>

²E.g. restrictions on the name of two elements linked by the ID/IDref mechanism can be enforced, but this would require a dedicated integrity-constraints checker like Schematron, on top of a grammar-based validator schema x[17].

³E.g. the commonality between inclusion and nesting representations.

4.1.1 Multistructured Data Models

Formally speaking, structured resources can be regarded as labelled graphs [122, 37]. From this point of view, the expressive limitations above can be regarded as a consequence of the fact that XML is based upon a restricted family of graphs that is inadequate for textual annotation, that is: trees. From this statement on, many formal or implemented models have been proposed over the last two decades; those propositions are generally rallied under the banners of Competing markup or Multistructured data models [141]. The term “multistructured” was proposed by reference to what *structure* means in XML (i.e. *trees*): a data model can be considered to be M-S provided well-formedness extends to graph models equal to or exceeding *forests*, that is graphs whose connected subgraphs are trees.

Precisely, it happens that the (pre-XML) CONCUR feature for SGML [85], which can be seen as the first historical M-S proposition, and one of the very few schema aware models, rest upon a forest formalism. In CONCUR, each element is indicated to be part of a given hierarchy, and several hierarchies are allowed on the same document; each of them can be validated against a dedicated DTD. The approach, however, has several shortcomings: mainly, no inter-DTD constraint can be expressed at all, and self overlap is problematic⁴. MuLaX [95] managed to transpose this philosophy in the XML world, despite an XML document can refer to one schema only. A MuLaX document mixes elements from several disjoint hierarchical paradigms, regardless of overlap; each hierarchy defines a projection, which yields a well-formed XML document that can be validated against an adequate XML schema; still, no constraint can be expressed between elements that do not belong to the same hierarchy. A (weak) solution to this problem was to be given in the MSXD model. Like MuLaX and CONCUR, MSXD is based upon the vision of a structure as a hierarchy. Unlike the two previous models, all the hierarchies are not instantiated in the same document: one XML document has to be written for each structure; and each document is validated separately by an appropriate RelaxNG schema. However, MSXD enables to express relationships between elements contained in separate hierarchies, by means of Allen’s relations [35, 34].

More advanced graph structures have also been proposed. Some focussed on well-bounded graph families. Significant examples of that movement are multitree-based TexMecs [97] or MCT model [101], in which the ascendant’s and the descendant’s graphs of any node are trees, and the restrained, acyclic polyarchy model GODDAG [162]. A clever grammar-based validation language, dubbed Rabbit/duck grammar, was proposed for TexMecs and similarly shaped documents [160]; it works on documents where annotation from competing hierarchies are entangled, and extracts the single hierarchies, which can be validated against a dedicated XML schema, but it does so while checking some inter-hierarchical constraints [174]. However, it falls short when it comes to validating polyarchies.

From this point on, many even more expressive models have been proposed, but rely on no particular graph model; most of them will also drop inline markup in favour of standoff markup. LMNL [175, 139] presents itself as the archetype of such models.

⁴Since DTDs validate only non overlapping elements, an arbitrary number of DTDs would be required to validate documents where multiple, self overlapping elements occur, as coined by [174]. This limitation applies to the two following approaches as well.

LMNL has a layered directed acyclic graph structure, which models elements as annotated ranges of text from a character stream [174]. Worth mentioning is the fact that LMNL enables to annotate annotations themselves, since they are no more than text streams themselves. Also, importantly, Creole, a prototype schema language, based on RelaxNG but extended to overlapping annotation, was tailored for LMNL [174]. SILL: LMNL [175] represents a more stripped-down vision of multilayer annotation. In many respects, LeAG borrows from LMNL. In LMNL, the user can identify *ranges* in a character stream and name them by means of pairs of opening and closing tags. Ranges themselves can be annotated by (meta)ranges, which inspired the attribute syntax in LeAG. Yet LMNL claims to be an *annotation* language solely, and not a *structuring* language: in particular, LMNL does not provide the user with means to represent inclusion or sibling relations. By sweeping out the notion of inclusion, LMNL seemingly clears the paradox out; yet LMNL is not absolutely blind to the charms of hierarchies: it lies upon the notion of ‘layers’, that is, ranges that fully contain the ranges that start and end in their scope, which is reminiscent of XML hierarchies – but if such patterns cannot be interpreted in structural terms, can they be but fortuitous patterns? Still, because hierarchies are a classic and fundamental annotation structure [184], the LMNL model comes along with XML generators that can extract hierarchies from the data. Our point, on that matter, is that since hierarchies are so central, the best is to enable the editors to have direct control over their expression – which indeed demands additional syntax. Apart from those critical considerations, LMNL is an important annotation model, that goes beyond most others, in terms of expressivity; moreover, it benefits from a grammar-based validation language [174], able to embrace the multilayer documents as a whole, which can be compared only to RDF validators (or to the SeAG we propose [15]).

Annotation Graphs [21, 19], that will be discussed thoroughly below is, from a formal point of view, a quite similar model; nonetheless, it adds an interesting notion of chronology, which can turn very handy for multimedia corpus annotation.

Eventually, several annotation models have originated from the RDF community. One may think of the pioneering RDFTef [178], the Open Annotation data model [150] or EARMARK [132]. The RDF data model, which imposes no restriction on the shape of the resulting graph, is very expressive; moreover, RDF annotation can be used as a complement to an existing TEI annotation [12], which is a way to ally the best of two worlds.

Figure 4.1 intends to capture the above diversity of models, classified in terms of the class of graph each model enables to express.

4.1.2 M-S Validation: Algorithmic Complexity

For the digital humanists, prominent user community of M-S documents, modelling resources by a schema, as a preliminary for annotation campaigns, is now part of the business [138]. Moreover, the existence of a schema offers many advantages in processing and querying structured data [79]. However, most validation mechanisms intended for M-S documents simply combine several XML schemas with rules to express constraints between the hierarchies each schema represents [85, 95, 152, 160, 35], providing the final user with a quite weak and clumsy modelling tool only. Said differently, few

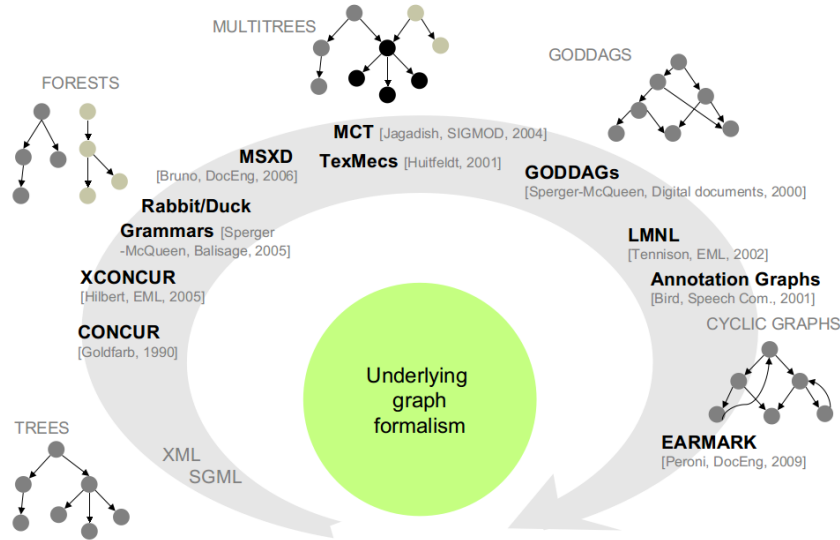


Figure 4.1: The classic multistructured data models, classified by the family of graphs of elements they enable to express.

schema models manage to get beyond multiple tree-structure checking.

A reason to that situation might be found in the algorithmic complexity of the current approaches adopted in M-S validation. It is generally assessed that the more expressive the data model, the higher the algorithmic complexity⁵ for the related processing tasks [79, 170, 126, 116, 112, 146]. This applies, at least, to grammar-based and rule-based [97] validation.

The tag “grammar-based” applies to the three main validation languages for XML, namely DTD, W3C XML Schema (XSD) and RelaxNG, which are commonly modelled as tree grammars, or tree automaton [126]. Although the schema languages do not recognize the same class of trees, it has been established that the tree languages that the three technologies enable to express are included, or equal⁶, fit like nesting dolls into regular tree languages, for which validation can be done in linear time in the documents’ size [112]⁷. However, as hinted by [170], there are tree grammars that may not even be decidable for the interpretation problem that is part of validation in RelaxNG and XSD. Integral rule-based validation with Schematron, as experienced by [17], does not overcome this trade-off: complexity is worse than with the grammar-based approach.

Few information is available about the time complexity of M-S validation mech-

⁵Following Murata’s assertion that time complexity matters more than space complexity, we will focus on the first only [126].

⁶DTD cover local tree languages, included in single-typed tree languages expressible by XSD, themselves included in regular tree languages that are covered by RelaxNG.

⁷The difference of expressiveness between the three schema languages comes at a cost though, but only for other processing tasks [79].

anisms. The complexity of MSXD’s validation mechanism, which theoretically combined RelaxNG validation for each independent tree structure with Allen’s relations to express inter-trees constraints, was in question in the first proposition paper [34]; actually, in the end, to the best of our understanding, it was not implemented as such [36]: Allen’s relations were rather translated into XQuery extensions and cross-hierarchies constraints could then be checked by assessing adequate queries on the data. Creole, the promising grammar-based schema language for MLNL that inherently understood overlap, and hence enabled to express competing hierarchies in the same document, as well as all structural constraints in a single schema, was prototyped using XSLT; despite the optimization techniques taken from RelaxNG, the result was considered “too slow” [174] – no other implementation followed, unfortunately.

It has to be mentioned here that RDF-based M-S data models suffer from the same trade-off. RDF, which is a very expressive model, is provided with an ontology language, OWL, as well as reasoners, which are sometimes used as validators [59]. However, using OWL to validate a document raises several difficulties and questions. First, the full version of OWL is undecidable, hence two main restrictions, OWL-DL and OWL-Lite, which still perform with repelling NEXPTIME and EXPTIME complexities [146]. Second, OWL axioms are not natively interpreted as integrity constraints by OWL reasoners, resulting in a rather weak validation mechanism [146], because of two main features of OWL that are the Open World Assumption and the No Unique Name feature, whose combination allows to assess when an assertion is verified, but not that is is not [159]. Indeed, techniques have been proposed to by-pass those limitations and enable the proper expression of identity constraints, but result in huge execution times [172]. Nonetheless, RDF validation is a promising field of research, as illustrated by the ShEx [143] and SHACL [107] projects. Time complexity still seems to be quite high, but cutting it down is being investigated [167].

4.2 The Annotation Graphs Model

The above enlightens a trade-off between expressiveness and complexity – trade-off that expresses, to be accurate, *inside the frame of a given validation technique*. For instance, while Brzozowski derivative-based validation [174] runs in linear time for regular tree languages, the same approach does not extend easily to more general graphs. This leads to question the use and tweak of XML and RDF tools for M-S validation, precisely because, as well engineered systems, they are already optimized for their native use.

Simulation [144, 37], is an interesting alternative to rule- and grammar-based descriptive formalisms. A simulation is a relation over (often rooted) directed labelled graphs. Informally, the existence of a rooted simulation of a graph B by a graph A implies that all the paths of B starting from its root have a matching path in A , whose label sequence is identical. Thus, A describes the structure of B , because all the patterns in B somehow have a match in A . Conversely, A behaves as a graph schema: it validates the graphs that contain only patterns defined in A , i.e. that A simulates.

Validation by simulation was first operated for semistructured (S-S) data [171, 3]. The Object Exchange Model (OEM) underlying S-S data is a cyclic, unordered, directed labelled graph. Natively, a S-S database is schemaless; Dataguides [86] or Graph Schemas [37] are inferred from the data. They are graphs that simulate the S-S database, providing a structural description that can be exploited for querying purposes. Simulation check performs in $O(|edges| \cdot |vertices|)$ [144], which is acceptably low for general graph-structured data validation.

Still, despite providing an expressive data model⁸ and an appropriate schema mechanism, as far as we know, S-S model was never tuned for annotation. Indeed, it lacks a clear representation of inclusion, a notion of order or a way to index nodes along reading dimensions to support linear annotation; moreover, since Dataguides and Graph Schemas are inferred from the data, they cannot be used as authoring tools.

Still, because the OEM is so general, the principle of a simulation-based validation is not restricted to S-S data but “can be applied easily to any graph-based data model” [86]. The eAG data model we propose in the Paragraph 5.1 is such a graph-based data model. It relies upon an important pre-existing data model, namely, Annotation graphs (AG), that appears to be one of the most generic and versatile annotation models from the late 1990’s. We propose to the reader a detailed, critical description of the historical AG model. We will point out its strength and weaknesses as an annotation model for DSE, so as to build, in its continuity, a similarly versatile model tailor-fit for scholarly annotation.

4.2.1 Annotation Graphs

Annotation Graphs (AG) [21] were introduced in the late 1990’s by Bird and Liberman as a generic model and language for annotation, at a moment when a plethora of competing models had been and were proposed. As evidence of the generic quality of the AG model, the authors showed that the state-of-the-art annotation languages, namely E-mu [41], LAF-GraF [98] or the SGML/TEI [161] could all be translated into AG without any loss of information, which gave AGs a certain fame among computer humanists and made it one of the most cited models of annotation. It was later implemented [82, 20] and a query language was experimented [19].

Even though AG was initially thought as a model for linguistic annotation, the flexibility of the model is such that it can be used for any resource that can be indexed, at the local scale⁹, along one dimension, hence making the AG model adequate, to a certain extent, for the annotation of multimedia resources (containing audio and text, typically).

Definition 4.2.1: Chronology, reference value. A **chronology** is a totally ordered set $\langle T_i, \leq_i \rangle$. An item from a chronology is called a **reference value**.

⁸Surprisingly, the OEM is referred to as “essentially equivalent to XML” on the Lore project website (infolab.stanford.edu/lore), which was a pioneering OEM DBMS before migrating to XML.

⁹See the notion of “chronology” – Definition 4.2.1, right below.

Example. A chronology for indexing a text stream can be built out of the inter-character positions in the text. A chronology for indexing an audio file can be built out of time stamps, in the appropriate granularity (ms, s, ...).

Definition 4.2.2: Annotation Graphs. An **Annotation Graph** AG over an alphabet \mathcal{L} and a collection of chronologies $\langle T_i, \leq_i \rangle_i$ is a tuple $(G = (V, E), ref, label)$, where

- $G = (V, E)$ is a directed, labelled graph,
- ref is a function $V \rightarrow \bigcup_i T_i$, that associates a reference value to the nodes of the graph G ,
- $label$ is a function $E \rightarrow \mathcal{L}$, that associates a label to the edges of the AG,

and so that :

1. for all node $v \in V$, $deg(v) \neq 0$;
2. for all oriented linear graph $v_1[\dots]v_2 \subseteq G$ connecting a node v_1 to a node v_2 , then if $ref(v_1)$ and $ref(v_2)$ are defined, there is a chronology $\langle T_i, \leq_i \rangle$ so that $ref(v_1) \leq_i ref(v_2)$.

Nota. According to the property 2. in the definition above, two nodes belonging to the same connected part of an AG cannot bear reference values belonging to distinct chronologies. Since the authors explicitly discard the possibility to compare reference values belonging to separate chronologies, one may neglect the index in the expression $ref(v_1) \leq_i ref(v_2)$ and simply write $ref(v_1) \leq ref(v_2)$.

Comment: Annotation paradigm. Annotating some content indexed along a given chronology $\langle T_i, \leq_i \rangle$ is done as follows: if the content is located between the reference values r_1 and r_2 in T_i , then the content is qualified by the label l iff the AG contains an edge labelled l between a node v_1 so that $ref(v_1) = r_1$ and a node v_2 so that $ref(v_2) = r_2$. Several such edges can be defined between a given pair of nodes. The annotated content is thus any portion of the resources delimited by a pair of reference values, while the way this content is annotated, or qualified, is determined by the label of the corresponding edge(s). In that, AG is a typical stand-off annotation model.

Noteworthy, AG shines by its ability to express multimedia annotation in a very natural way. Indeed, since several chronologies can be defined for the same AG, it is possible to define a chronology for indexing textual resources and another chronology for audio files, and then to annotate a multimedia resource mixing text and audio by means of a unique graph of annotation.

Nota. Even though an AG is represented by a unique graph, the AG model does not require this graph to be connected: it may on the contrary be constituted of several connected parts.

In particular, this permits to leave certain portions of the resources free of any annotation. Also, two distinct connected subgraphs of the AG can overlap freely. As a consequence, AG support the expression of a wide range of annotations: annotation with gaps; overlapping annotation; multilayer annotation; annotation of moments (i.e. resources wholly positioned at a certain reference value along a certain chronology)... see Figure 4.2 page 69.

Definition 4.2.3 : Structural and referential orders. Be $(G = (V, E), ref, label)$ an AG. Two partial order relation can be defined on the AG:

1. **Structural order \leq_s .** Be $v_1, v_2 \in V$. Let us define the s-precedence \leq_s by:

$$v_1 \leq_s v_2 \Leftrightarrow \exists J \in \mathbb{N}^*, \exists \{e_j\}_{j \in [0; J]} \mid v_1[e_1 \dots]v_2.$$

2. **Referential order \leq_r .** be $v_1, v_2 \in V$. Let us define the r-precedence \leq_r by:
 $v_1 \leq_r v_2 \Leftrightarrow ref(v_1), ref(v_2)$ are defined, belong to the same chronology and verify $ref(v_1) \leq ref(v_2)$.

Definition 4.2.4 : Precedence. Be $(G = (V, E), ref, label)$ an AG. **Precedence** is defined as the transitive closure of the union of the s- and the r- precedences:

$$\forall (v, v') \in V, v < v' \Leftrightarrow \exists n \in \mathbb{N}^*, \exists \{v_0 \dots v_n\} \in V^{n+1} \mid v_0 = v, v_n = v' \\ \text{and } \forall i, v_i <_s v_{i+1} \vee v_i <_r v_{i+1}$$

Comment As a consequence of the Definition 4.2.2 above, in a connected part of an AG, the structural and the referential order match, for the nodes for which they are both defined.

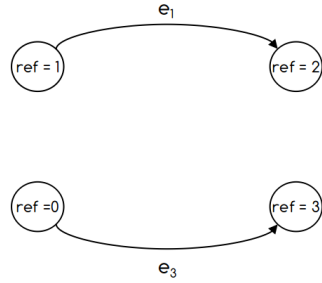
This property enables to define a notion of inclusion between edges as follows:

Definition 4.2.5 : Inclusion. Be $(G = (V, E), ref, label)$ an AG. Be the following graphs, included in the AG : $v_1[e]v_4$ et $v_2[e']v_3$.

The three following partial relations can be defined:

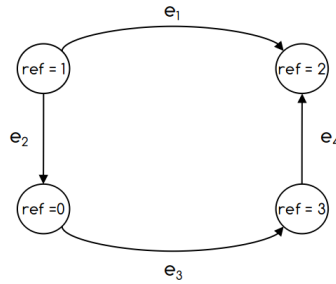
- **Structural inclusion \subset_s .** $e \subset_s e' \Leftrightarrow v_1 \leq_s v_2 \wedge v_4 \geq_s v_3$.
- **Referential inclusion \subset_r .** $e \subset_r e' \Leftrightarrow v_1 \leq_r v_2 \wedge v_4 \geq_r v_3$.
- **Inclusion \subset .** Inclusion is the transitive closure of the union of the structural and referential inclusions.

Example. Be the following AG:



In this graph, $e_1 \subset e_3$.

Comment. The fact s-precedence and r-precedence shall match on the nodes from a connected part of an AG enables to discard annotation patterns in which, otherwise, inclusion would be paradoxical. For instance, the following graph (which is not a correct AG), the above definition of inclusion would give: $e_3 \subset_s e_1 \wedge e_1 \subset_r e_3$:



However, a crucial drawback from imposing the match between s- and r- precedence is that AG cannot contain cycles, while cycles can be a good means to represent arbitrary relations and links.

Definition 4.2.6: Domination. Be $(G = (V, E), ref, label)$ an AG.

An edge $e \in E$ is said to **dominate** a sequence of edges $[e_i]_{i \in [1;N]} \in E^N$, $N \geq 0$ iff $\exists (v, v') \in V^2$, $[v_i]_{i \in [1;N+1]} \in V^{N+1}$ so that:

- $v[e]v' \subset G$, $v_1[e_1] \dots [e_N]v_{N+1} \subset G$
- et $v_1 = v$, $v_{N+1} = v'$.

Definition 4.2.7 : Structural hierarchy. Domination is how hierarchies are described in the AG model. Indeed, all the edges of a sequence of edges that is dominated by an edge e are included in e .

Definition 4.2.8 : N-ary relations between edges. On top of the above structural considerations, AG enable the definition of n-ary relations between edges, by means of optional fields in the labels.

- **Annotation layers, types.** An **annotation layer** is any set of edges possessing the same type. The type of an edge is characterized by the value of a prefix added to the edge's label, ending by the delimiter '/'.
- **Equivalence classes.** Edges belonging to distinct layers can be grouped into **equivalence classes**. The equivalence class is defined by adding an identifier as a suffix to the label of those edges. The suffix starts by the delimiter '/'.
- *Structural dependencies.* **Structural dependencies** enable to assert that one or several edges are related to a given edge. To define a structural dependency, a suffix ':N/M', is added to the edges, $N, M \in \mathbb{N} \cup \{'\}$, where N identifies the current edge and M identifies the edge it is related to. The value '-' serves as a blank value.

Comment. An annotation layer is a very open notion, from a structural point of view – no less, actually, than any set of annotation layers ([21] p. 40), since it is solely a collection of edges possessing the same type. Figure 4.2 illustrates the possible annotation configurations inside an annotation layer.

The other kinds of relations – the structural dependencies in particular – aim at enabling the expression of arbitrary relations that could otherwise have been represented by cycles.

Attributes in AG... The authors do not define any syntax for refining the labels with what could be called attributes, that is, additional type-value data. They only state that the label can be freely structured into several fields in order to do so, but leave the realisation of that prospect to the user...

The resulting AG model is, in the end, highly expressive, as evidenced in the original article [21], enabling the expression of overlapping, multilayer annotations, as well as arbitrary N-ary relations between the edges of the AG, whose role in the annotation is reminiscent of XML elements.

However, the adequateness of the syntactical solutions chosen by the authors to reach this expressivity can be questioned. In the following, we propose several amendments to this initial model, based on the criticism of the AG syntax, and define extended Annotation Graphs.

4.2.2 Criticism and Necessary Amendments to the Annotation Graphs

Our criticism focus on two main points:

1. The first point regards the fact an AG may be a disconnected graph, and why. AG offer the possibility to define several chronologies is thought as a means to enable the annotation of resources from different media types (e.g. text vs. audio vs. images...). Still, as stated above, connected parts in an AC may only

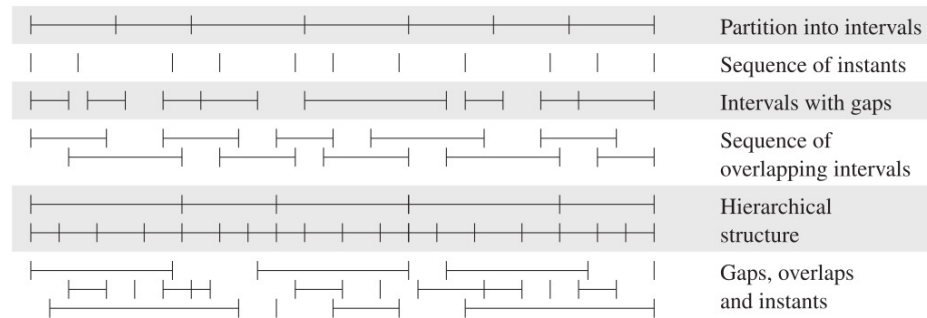


Figure 4.2: Possible configurations of an annotation layer in the Annotation graph model (extracted from [21], p. 40.)

contain nodes that bear reference values from *the same* chronology. Thus, an annotation requiring several chronologies will result in a graph containing at least two disconnected subgraphs. Yet, instead of gathering together nodes and edges making use of the same chronology, i.e. involved in the annotation of the same type of content, it seems more natural that connected parts of an annotation graph shall gather the nodes and edges involved in the annotation of *the same resource*, be it multimedia.

Consider a born-digital document containing an introductory paragraph, an audio file and then a textual comment of the audio file. In the AG model, the two textual paragraphs will have to be annotated by means of a separate connected graph compared to the audio file; even worse, it would be impossible to express the fact that the audio file is positioned between the two paragraphs, that is, that three items of different nature (text, audio and text) form a sequence and are included in a unique document.

One of the main amendments eAG will operate compared to AG will be to permit the coexistence of several chronologies in the same connected part of the annotation graph – to the point that having separate connected parts will be useless, so well so that the eAG graph model will be a connected graph. To achieve this goal, we will introduce the notion of composite chronology and the notion of epsilon edge for expressing ‘blank annotations’, to keep the interesting ability AGs offer not to annotate all the primary content.

2. The second questions the fact that information that is not, in substance, dissimilar, is not represented homogeneously in AG. In particular, the fact that a text, if it is part of the primary corpus (e.g. the literary documents to be edited), will possibly be described in great detail by a set of edges and nodes (i.e. an AG annotation), while if it is part of the critical apparatus (i.e. if it is the label of an edge), will possibly be described further only by means of interpretable fields added to it as suffixes and prefixes (links and attributes as defined in the AG model), i.e. by means of chains of characters. Also, the syntax for representing structural precedence, which is a binary relation between edges, is structural in the sense it involves nodes and edges while semantic relations (e.g. structural

dependencies) are simply represented by means of identifiers. Our answer to solve this heterogeneity is twofold:

- Based on the consideration that the attributes of an element of annotation are structured data relative this element, we propose to model attributes as an annotation on top of an annotation. In that, we follow the example of LMNL [175].
- While the notion of structural dependencies are a convenient means to represent relations while avoiding cycles in the AG, and thus to maintain the structural and referential orders consistent across the graph, we propose to introduce a distinctive syntax for the label of the edges that involve cycles, i.e. that represent semantic relations between annotation elements. This mechanism enables to maintain the consistency of the structural and referential orders in the graph stripped from those special edges, while offering a homogeneous, structural representation for all relations.

We hereby define the corresponding eAG model in detail.

Chapter 5

Extended Annotation Graphs and Schema models

5.1 The Extended Annotation Graphs Model

We introduce here Extended Annotation Graphs (eAG), a data model derived from Annotation Graphs. The eAG syntax is first introduced informally on a textual annotation example¹, that will serve as a running example over this part and the next one. Then, a formal definition of eAG is proposed, illustrated by means of a toy document mixing text and image.

5.1.1 An Example of eAG Annotation: Anaphoric Chains

A common linguistic annotation is the identification of anaphoric chains³ (AC). ACs are sequences of singular expressions so that if one of them refers to something, then they all do [44]. Consider the text given on Figure 5.1, adapted from *The Village of Ben Suc* by J. Schell. In this text, one may identify, among others, the following ACs: [a young prisoner / he / him / him / the prisoner], [An American observer who saw the beating], [the beating that happened then / the beating].

Annotating the text in terms of ACs made out of *expressions* is not trivial in XML. Since ACs do not form neither a sequence nor a hierarchy, they cannot be represented as normal, spanning XML elements. The classic solution is to identify only the singular expressions in the text and then relate them together accordingly by their IDs in `<linkgrp>` elements [55]. That solution, apart from being hard to

¹Additional examples of eAG annotations can be found elsewhere in the document. The Appendix11 (in French) offers an annotation of the Voynich manuscript², that makes intense use of the notion of composite chronology defined below as a means to encode the continuous quality of a text that is disseminated across the folio in variously shaped graphical containers. The notion of *quotes*, that is introduced in this part, is also illustrated in the next part, in Paragraph 6.2.4.

³The assisted, or automatic, identification of Anaphoric chains is an actively investigated field of research in NLP [140]; yet our purpose does not consider the process of identifying ACs, but how to represent, by means of an annotation language, the result of this identification, in the track of [55].

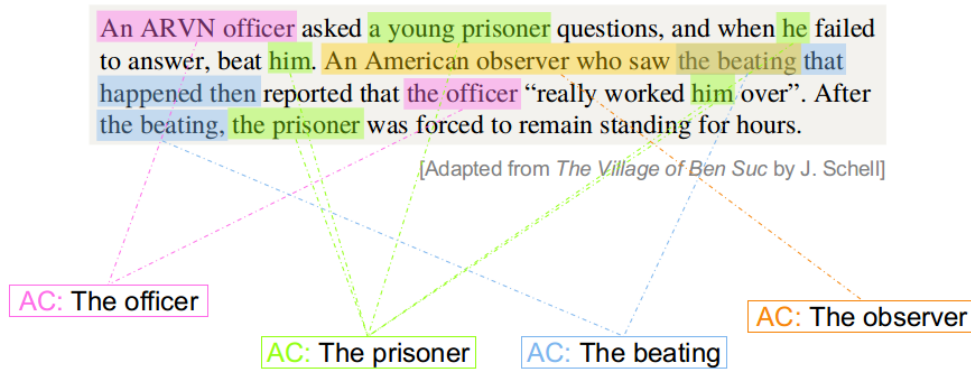
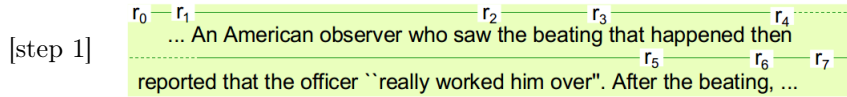


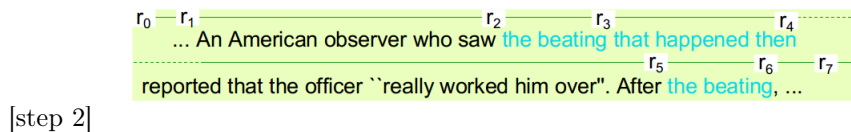
Figure 5.1: An adapted passage from *The Village of Ben Suc* by J. Schell, with some highlighted anaphoric chains and constituting singular expressions.

validate, does not represent the fact an AC is *composed of* expressions consistently with the XML syntax. Moreover, it does not extend to this example, which exhibits self-overlap [162]⁴.

In eAG, annotating anaphoric chains is straightforward. First, a chronology, indexing the text stream, must be defined, e.g. as a set of inter-character positions:



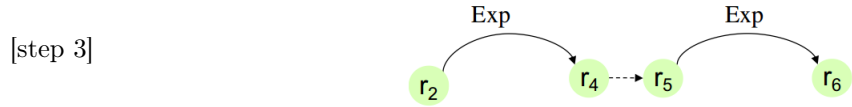
The encoding of the two singular expressions *Exp* regarding the beating is done as it would be in AG:



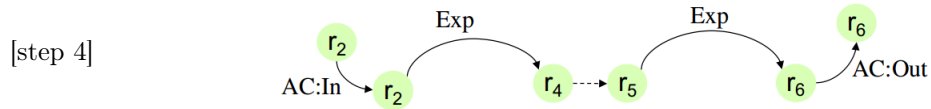
To model the fact that the two above expressions form a *sequence*, as suggested by the definition of anaphoric chains above, and that this sequence is discontinuous (in that the second *Exp* does not start where the first ends), we introduce a special edge for blank annotation, called the ϵ -edge⁵. It is operated as follows:

⁴Cf. *An American observer who saw the beating* and *the beating that happened then*.

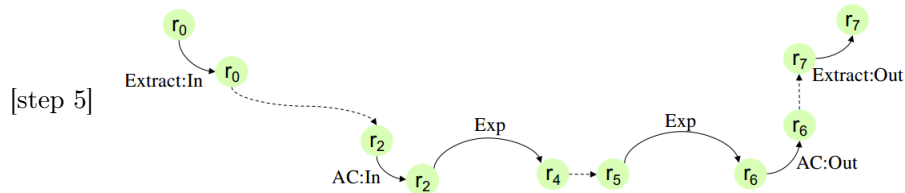
⁵The inspiration for this edge comes from the homonym ϵ -edges from the Finite State Machine theory, which denote transitions that do not consume any input character. The rationale for this loan is explicated in Paragraph ??



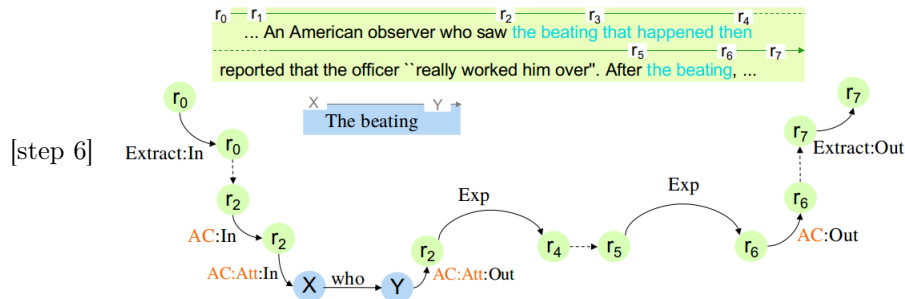
Now assessing that this sequence constitutes an anaphoric chain can be done by defining an *AC* element, for Anaphoric Chain, so that the sequence of *Exp* is included in *AC*. Inclusion, in eAG, is done by means of a pair of opening and closing edges, labelled with the name of the container element (*AC* here) and the suffixes *:In* and *:Out* respectively, as follows:



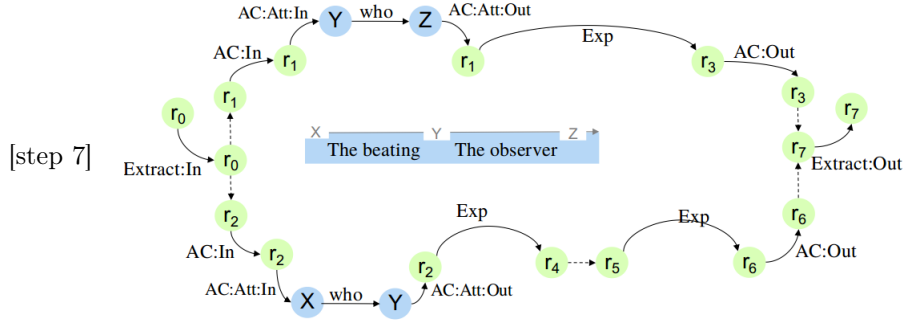
Assessing that the *AC* takes place inside an Extract is done similarly:



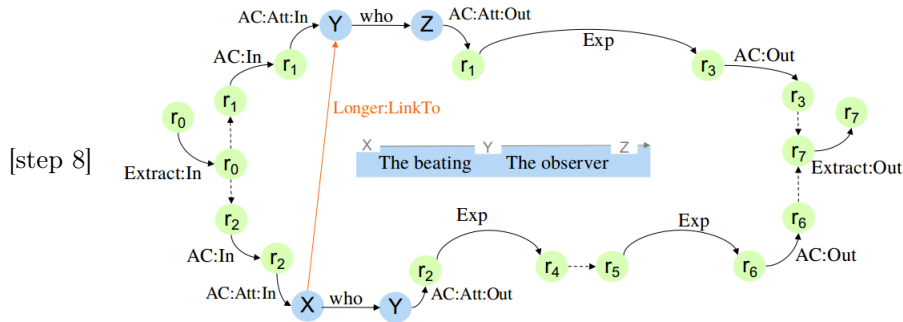
Adding attributes to an element whose name is *X* is done by inserting an element named *X:Att* inside *X*, the element *X:Att* in turn containing a small eAG indexing some added secondary resource, that corresponds to the content of the attributes. The following step provides a name “who” to the anaphoric chain, indicating that the AC is related to “the beating”:



Identifying a second, overlapping *AC* element (the one regarding the observer for instance) can be done just as previously, independently from any previous annotation (in particular, without worrying about overlapping elements):



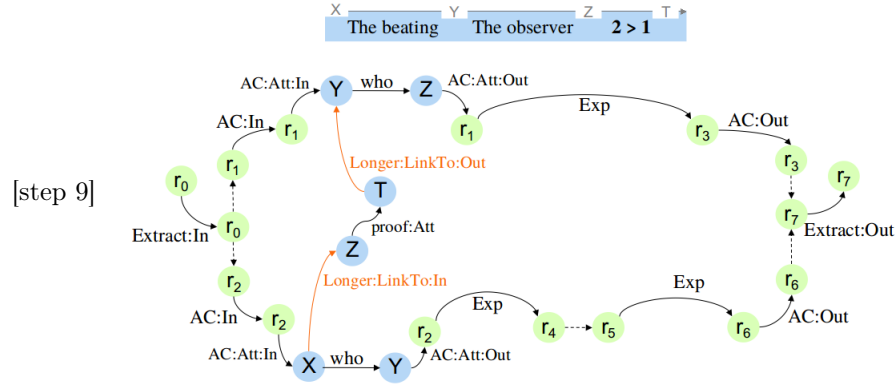
Eventually, suppose we aim at identifying the ACs and their constitutive expressions, but also to qualify their relative weight, e.g. by reifying the relation ‘this AC contains more expressions than that one’. The reification of that relation can be done by means of an edge bearing a special suffix `:LinkTo`, originating in the attributes of AC relative to *the beating* and targeting the other AC as follows:



This last graph ([step 8]) is a proper, or well-formed, eAG⁶, making use of the notions of element, of attributes, of hierarchy of elements and of semantic relations, or links. It represents a bi-layer annotation exhibiting self overlap.

This informal presentation of eAG shows several notions that the original AG failed to capture conveniently: for instance, the fact that two expressions form a sequence with a gap would have been represented by two disconnected edges assembled into a layer of annotation – while the fact two elements form a gapless sequence is represented differently, by the communality of the leaf of the first element with the root of the second; here, any kind of sequentiality relation is represented homogeneously, that is, structurally. Similarly, the semantic relation *Longer* is structurally represented here, while it would have demanded the use of the notion of structural dependency in AG. Representing relations by means of nodes and edges is not only satisfying because it provides a harmonious syntax. It also opens to the possibility to structure relations further, for example to include, in the linking element, some critical comment. For instance, the graph below provides an elaborate proof of the fact that the AC regarding the beating is “longer” that the AC regarding the observer, based on the number of Exp each contain:

⁶Noteworthy, the graphs given on step 3 to 7 are also well formed. The graph on step 2 is not, since it is not connected – see Paragraph 5.1.2.1.



As will be shown in Paragraph 5.2, it also enables to validate the relations properly, by restricting the nature of the elements that can be connected together by means of a certain relation.

5.1.2 The eAG Model, Formally

After this example-based, informal introduction, that aimed at providing the reader with a concrete representation of the eAG syntax – which may make the following more comfortable to read –, we now define the eAG model formally. We gradually introduce the notion of composite chronologies, that permit to handle documents that mix several types of information (text, image, etc.); we then present the formal model for hierarchies, elements, attributes, comments and links, together with the properties those features have to verify. Eventually, we discuss the notion of elements in eAG in a way that hopefully illustrates the expressive power of the model.

Throughout this formal presentation, the toy document represented in Figure 5.2 will serve to illustrate eAG's expressive power. It is constituted of one paragraph spanning over two pages, whose text locally refers to parts of a pictorial figure. Additionally, the pages are structured into modules, a module being the longest vertical unit containing data of homogeneous nature (viz. text vs. images here). The pictorial figure itself, *accidentally*, nests inside the paragraph (without being *part* of it).

5.1.2.1 eAG Graph Model

Extended Annotation Graphs are based on a cyclic graph formalism, as follows.

Definition 5.1.1: eAG graph model. An **extended Annotation Graph** G is a tuple $(G = (V, E), ref, label)$, where G is rooted, single-leafed and connected:

1. (**rooted**) $\exists v_r \in V$ so that $root(G) = \{v_r\}$
2. (**single-leafed**) $\exists v_l \in V$ so that $leaf(G) = \{v_l\}$
3. (**connected**) $\forall v \in V$, $\exists \varphi_v$ a path verifying $\varphi_v \subseteq_v^{root(G)} G$

The functions $label$ and ref are governed by the definitions and properties that follow.

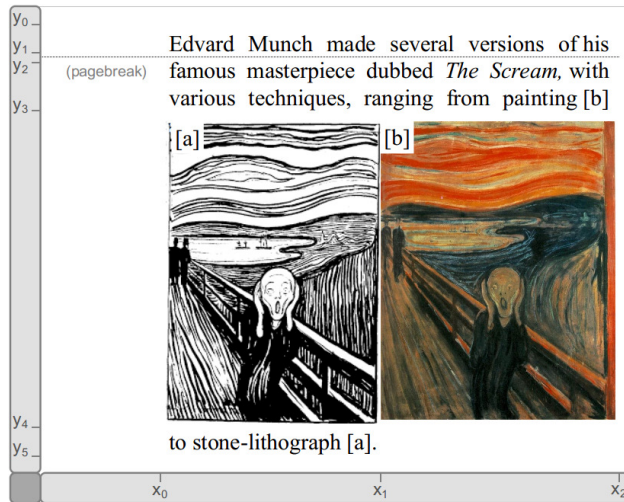


Figure 5.2: Document showing overlap, a figure enclosed in text and internal references.

5.1.2.2 Authorized Labels in eAG

The labels in eAG are not indifferent. We provide them with a minimal semantics, in order to represent the modelling notions of inclusion, attribute, links, etc.

Definition 5.1.2 (Labels) Be a special character ϵ ⁷. Be \mathcal{L}_\emptyset , $\epsilon \notin \mathcal{L}_\emptyset$, a set of strings that do not contain the character “:”. $\mathcal{L}_0 = \mathcal{L}_\emptyset \cup \{\epsilon\}$ is the set of unsuffixed labels. Additionally, be $\mathcal{S} = \{:\text{In}, :\text{Out}, :\text{Att}, :\text{Com}, :\text{LinkTo}\}$ the set of suffixes. Labels can be iteratively suffixed according to the rules given in Table 5.1. Those rules also define classes of labels, e.g. \mathcal{L}_{In} the set of labels whose last suffix is $:\text{In}$. The set of all labels \mathcal{L} is the union of all the preceding classes.

Nota. Noteworthy, Table 5.1 implies that an attribute can have attributes itself, and that a link cannot have an attribute – see the next paragraph.

Notation. In the following, given two strings l and s , $s \subset l$ denotes the fact that l contains the substring s .

5.1.2.3 Chronologies in eAG

Extending a notion introduced in the AG data model, we propose a new definition of chronologies, that is how locations in composite resources can be made reference to.

⁷ ϵ stands for a blank, or void, annotation.

Table 5.1: Allowed suffixes per label class and the resulting class. “-” stands for “undefined”.

$\varphi \rightarrow$ suffixed by	\emptyset	:In	:Out	:Att	:Out	:LinkTo
$l \in \mathcal{L}_{\emptyset}$	\mathcal{L}_{\emptyset}	\mathcal{L}_{In}	\mathcal{L}_{Out}	\mathcal{L}_{Att}	\mathcal{L}_{Com}	\mathcal{L}_{LinkTo}
$l \in \mathcal{L}_{In}$	\mathcal{L}_{In}	-	-	-	-	-
$l \in \mathcal{L}_{Out}$	\mathcal{L}_{Out}	-	-	-	-	-
$l \in \mathcal{L}_{Att}$	\mathcal{L}_{Att}	\mathcal{L}_{In}	\mathcal{L}_{Out}	\mathcal{L}_{Att}	-	-
$l \in \mathcal{L}_{Com}$	\mathcal{L}_{Com}	\mathcal{L}_{In}	\mathcal{L}_{Out}	\mathcal{L}_{Att}	-	-
$l \in \mathcal{L}_{LinkTo}$	\mathcal{L}_{LinkTo}	\mathcal{L}_{In}	\mathcal{L}_{Out}	-	-	-

Definition 5.1.3 A (general) chronology is any ordered set $\langle T, \leq \rangle$. Be then \mathcal{C} a set of strings called “chronometer names”. Be $m \in \mathcal{C}$. The reference space associated to m is a unique ordered set $\langle \mathbb{T}_m, \leq_m \rangle$. A chronology over $m \in \mathcal{C}$ is an ordered set $\langle T, \leq_m \rangle$ so that $T \subseteq \mathbb{T}_m$.

Illustration part 1. The document illustrated on Figure 5.2 contains text and a figure, both encapsulated into modules. The figure contains several images displayed side by side.

Several chronometer names can be defined: 1. *text* for indexing text in the reading order, line by line, 2. *vpos* for delimiting text blocks and figures along the vertical dimension across the different pages, 3. *hpos_{fig}* for delimiting images along the horizontal dimension in the figure denoted *fig* here.

Appropriate reference spaces for those chronometer names and this document are the following:

1. (*text*) $\mathbb{T}_{text} = \{0, 40, 41, 83, 84, 129, 130, 154\}$, i.e. the set of inter-character positions preceding (resp. following) each first (and last, resp.) character of each line (linebreaks counting for one character)⁸.
2. (*vpos*) $\mathbb{T}_{vpos} = \{y_0, y_1, y_2, y_3, y_4, y_5\}$, i.e. the set of vertical positions of the beginning and the end of each module of each page of the document, considering that each page follows the previous one along that vertical dimension.
3. (*hpos*) $\mathbb{T}_{hpos_{fig}} = \{x_0, x_1, x_2\}$, i.e. the set of horizontal positions serving as the border for an image in the only figure of the document.

Several chronologies can then be defined over those chronometer names and reference spaces:

1. $\langle T_{y,1}, \leq_y \rangle$ over the chronometer name *vpos*, with $T_y = \{y_0, y_1\}$, for the delimitation of the modules in the first page,
2. $\langle T_{y,2}, \leq_y \rangle$ over the chronometer name *vpos*, with $T_y = \{y_2, y_3, y_4, y_5\}$, for the delimitation of the modules in the first page,

⁸The first line starts at character 0 and ends at character 40, the second line starts at character 41, etc.

3. $\langle T_x, \leq_x \rangle$ over $hpos$, with $T_x = \mathbb{T}_{xposfig}$, where those values enable to delimit the two images in the pictorial figure,
4. $\langle T_c, \leq_c \rangle$ over $text$, with $T_x = \mathbb{T}_{text}$.

Comment. The use of chronometer names enables to define several chronologies on the same reference space, and thus, in the end, to compare values from different chronometers, when appropriate. See the following definition and Example 5.1.1 below.

Definition 5.1.4 (Concatenation) Be $\langle T_a, \leq_a \rangle$ and $\langle T_b, \leq_b \rangle$ chronologies. $\langle T_a \cdot T_b, \leq_{a,b} \rangle$ defines a chronology over $T_a \cup T_b$ iff the following relation $\leq_{a,b}$ defines an order over $T_a \cup T_b$:

For any $t, t' \in T_a \cup T_b$, then:

- $t =_{a,b} t' \Leftrightarrow \exists x \in \mathcal{C} \mid (t, t') \in \mathbb{T}_x^2 \wedge t =_x t'$
- $t <_{a,b} t' \Leftrightarrow \exists x \in \mathcal{C} \mid (t, t') \in \mathbb{T}_x^2 \wedge t \leq_x t'$
or $\nexists x \in \mathcal{C} \mid (t, t') \in \mathbb{T}_x^2 \wedge (t, t') \in T_a \times T_b$.

Property 5.1.1 Be three chronologies $\langle T_a, \leq_a \rangle$, $\langle T_b, \leq_b \rangle$ and $\langle T_c, \leq_c \rangle$ so that $\langle T_a \cdot T_b, \leq_{a,b} \rangle$ and $\langle T_b \cdot T_c, \leq_{b,c} \rangle$ are defined. Then:

$$\langle (T_a \cdot T_b) \cdot T_c, \leq_{(a,b),c} \rangle \text{ is defined } \Leftrightarrow \langle T_a \cdot (T_b \cdot T_c), \leq_{a,(b,c)} \rangle \text{ is defined.}$$

If defined, then, $\langle (T_a \cdot T_b) \cdot T_c, \leq_{(a,b),c} \rangle$ and $\langle T_a \cdot (T_b \cdot T_c), \leq_{a,(b,c)} \rangle$ will be denoted $\langle T_a \cdot T_b \cdot T_c, \leq_{a,b,c} \rangle$ indifferently.

Example 5.1.1 Be two chronometer names nat and ab associated to the reference spaces \mathbb{N} and 2^Λ , where Λ is the roman alphabet of letters, and the natural order \leq_{nat} and the alphabetical order \leq_{ab} respectively.

Be $\langle T_1 = \{0, 1, 2\}, \leq_1 \equiv \leq_{nat} \rangle$, $\langle T_3 = \{2\}, \leq_3 \equiv \leq_{nat} \rangle$ and $\langle T_4 = \{4\}, \leq_4 \equiv \leq_{nat} \rangle$ three chronologies on nat . Be $\langle T_1 = \{X, Y\}, \leq_2 \equiv \leq_{ab} \rangle$ a chronology on ab .

$\langle T_1 \cdot T_2, \leq_{1,2} \rangle$ defines a chronology, where $2 <_{1,2} X$ for instance.

$\langle T_1 \cdot T_2 \cdot T_3, \leq_{1,2,3} \rangle$ does not define a chronology (the antisymmetry would not hold), while $\langle T_1 \cdot T_2 \cdot T_4, \leq_{1,2,4} \rangle$ does.

Definition 5.1.5 (Inclusion) Be $\langle T_a, \leq_a \rangle$ and $\langle T_b, \leq_b \rangle$ chronologies.

We say $\langle T_b, \leq_b \rangle \subseteq \langle T_a, \leq_a \rangle$ iff $\exists (T_1, T_2) \subset T_a^2$ so that $\langle T_1 \cdot T_b \cdot T_2, \leq_{a,b,a} \rangle$ defines a chronology.

Example 5.1.2 In Example 1, $\langle T_2, \leq_2 \rangle \subseteq \langle T_1 \cdot T_4, \leq_{1,4} \rangle$.

This notion of chronology permits to index a composite, yet continuous content.

Illustration, part 2. Consider the second page in Figure 5.2. As stated before, it contains three modules, containing two text lines, a figure, and one line respectively. Following *Illustration part 1*, three chronologies can be defined for the annotation of the second page: $\langle T_y, \leq_y \rangle$ over the chronometer name $vpos$, with $T_y = \{y_2, y_3, y_4, y_5\}$ (in ascending order), for the three modules delimitation; $\langle T_x, \leq_x \rangle$ over $hpos$, with

$T_x = \{x_0, x_1, x_2\}$, for the figure decomposition into images; $\langle T_c, \leq_c \rangle$ over *text*, with $T_c = \{41, 83, 84, 129, 130, 154\}$ ⁹, based on characters (including linebreaks) count, for lines indexation.

By double inclusion, we can define a chronology $\langle T, \leq \rangle$ over $T_c \cup T_x \cup T_y$, so that $y_2 < 41 < 83 < 84 < 129 < y_3 < x_0 < x_1 < x_2 < y_4 < 130 < 154 < y_5$. See Figure 5.3 in Paragraph 5.3 for an annotation of the second page of the document making use of this chronology.

Definition 5.1.6 (References) In an eAG $G = ((V, E), ref, label)$, a reference $ref(v)$ is associated to each node $v \in V$. For each v , there is a unique reference space $\langle \mathbb{T}_c, \leq \rangle_c$ so that $ref(v) \in \mathbb{T}_c$.

Definition 5.1.7 (Range) Two references r_1 and r_2 belonging to the same chronology and sharing the same reference space identify a range within the resources, that can be annotated.

Comment. Ranges are then annotated by defining either one or more hierarchies of elements of the kind that was informally introduced in Paragraph 5.1.1, whose first node(s) bears the reference value r_1 and whose last node bears r_2 .

Property 5.1.2 Be $G = (V, E)$ an eAG, $v, v' \in V, e, e' \in E$ so that $v[e], v[e'] \subseteq G$. Then $label(e) = label(e') \Rightarrow e = e'$.

Comment. This property discards eAGs in which two nodes shall be connected by two edges with the same label, obvious case of redundant annotation (in which the same range is annotated twice with the same label).

We now define elements, hierarchies of elements, and the other useful structures formally.

5.1.2.4 Elements, Hierarchies and Links in eAG

eAG rest upon a notion of elements that is not unrelated to the now classic notion of XML elements. One major difference though is that in eAG, elements are of four different kinds, according to the class of labels their name belongs to, as indicated in Table 5.2.

Structurally speaking, elements are defined as below.

Definition 5.1.8 (h-equality and dominance). Be $G = (V, E)$ a graph, and be $(\{v_0 \dots v_N\}, \{e_0 \dots e_{N-1}\})$ a path included in G . Be $n, m \in \mathbb{N} ; 0 \leq n \leq m \leq N$. We define h-equality (denoted $=^h$), h-dominance (denoted $>^h$) and border-h-dominance (denoted $>_b^h$) as follows.

v_n is said to be h-equal to v_m , denoted $v_n =^h v_m$, iff :

1. $n = m$ OR

⁹The first line of page 2 starts at character 41, etc.

label class of the element name	kind of element
\mathcal{L}_\emptyset	Elementary spanning elements
\mathcal{L}_{Att}	Attributes
\mathcal{L}_{Com}	Comment
\mathcal{L}_{LinkTo}	Links

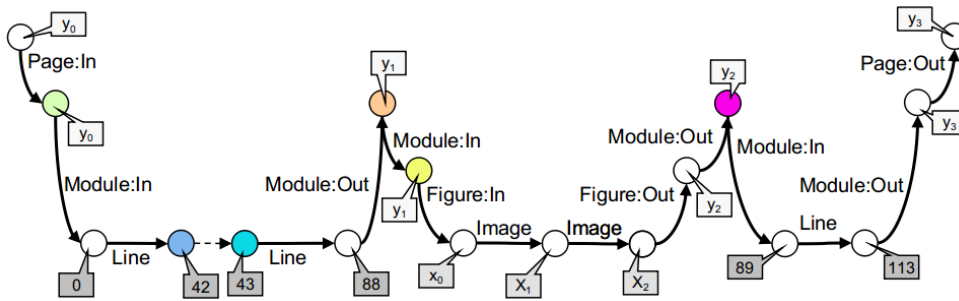
Table 5.2: The different kinds of eAG elements, according to the label class their name belongs to.

2. $\forall j \in [n, m-1], label(e_j) \in \mathcal{L}_\emptyset \cup \mathcal{L}_{Att} \cup \mathcal{L}_{Com}$ OR
3. $(v_n \not\geq^h v_m \wedge v_m \not\geq^h v_n)$ AND $\{\forall k, l; n < k \leq l < m, v_n \geq^h v_k \wedge v_n \geq^h v_l \wedge v_m \geq^h v_k \wedge v_m \geq^h v_l$ AND $[v_k =^h v_l \vee (v_k >^h v_l \vee v_l >^h v_k)]\}$.
- When $n \neq m$, v_n and v_m are said to border-h-dominate the nodes $v_i, i \in [n+1, m-1]$, denoted $(v_n, v_m) >_b^h v_i$, iff :
- a. $\exists l \in \mathcal{L}; label(e_n) = l:In$ and $label(e_{m-1}) = l:Out$ and
b. $\forall j < k \in [n+1, m-1], v_j =^h v_k$
or $\exists(x, y) \in [j, k-1] \times [k+1, m-1]; (v_x, v_y) >_b^h v_k \wedge v_j =^h v_x$
or $\exists(x, y) \in [n+1, j-1] \times [j+1, k]; (v_x, v_y) >_b^h v_j \wedge v_k =^h v_y$.
- Be $A, B \in [0, N]$. v_A h-dominates v_B , denoted $v_A >^h v_B$, iff $\exists n, m \in \mathbb{N}; 0 \leq n \leq m \leq N \mid (v_n, v_m) >_b^h v_B \wedge v_A =^h v_n$.

Illustration part 3-b. Let us consider the annotation of the second page of the document represented on Figure 5.2 page 76 as follows: it is one *Page*, containing three *Modules*. The first *Module* contains two *Lines*; the second *Module* contains a *Figure*; the last *Module* contains one *Line*. Additionally, the *Figure* contains two *Images*.

The chronology we may use for this annotation has been defined in *Illustration part 2*: it is $\langle T, \leq \rangle$ over $T_c \cup T_x \cup T_y$, so that $y_2 < 41 < 83 < 84 < 129 < y_3 < x_0 < x_1 < x_2 < y_4 < 130 < 154 < y_5$.

The above hierarchical¹⁰ description translates into the following eAG:



¹⁰It would also be possible to edit the document by focussing on the text, by annotating it, in an additional hierarchical layer, in terms of *Paragraphs* containing *Lines*, or *Paragraphs* containing internal *References*, for instance, as illustrated in Figure 5.3.

The blue nodes a h-equal. The green, the pink and the orange nodes are also h-equal. The green and the orange nodes border-h-dominate the blue nodes. The pink node h-dominates the blue nodes. Noteworthily, the blue nodes are not h-equal to the yellow node.

Property 5.1.3 a) As a consequence of the point 2. in Definition 5.1.8, $\forall v[e]v' \subseteq G$:
 “:LinkTo” \subset $label(e) \Rightarrow v \neq^h v' \wedge v \not\prec^h v' \wedge v' \not\prec^h v$.
 b) We impose that the reciprocal shall be true.

Comment. Point b) means that two nodes from an eAG that are connected by an edge will have a hierarchical relationship, apart from the case the edge that connects them participates in defining a link, which suggests that eAGs will be substantially made out of (interlaced, possibly overlapping) hierarchical patterns. Point a) is absolutely crucial in the eAG model. It means that two nodes separated by an edge participating in defining a linking element cannot be compared in hierarchical terms. See Property 5.1.8 and the subsequent comment for the practical consequences of that fact.

Definition 5.1.9 (h-levels) Be an eAG G . Since G is connected, $\forall v \in V, \exists P = (V_P, E_P) \subseteq G$ a root-to-leaf path so that $v \in V_P$. The h-level of v in P is the biggest subset $N \subseteq V_P$ so that $\forall v' \in N, v' =^h v$.

(h-levels direct inclusion) Be a path P , N_x, N_y h-levels in P . N_y is directly included in N_x , denoted $N_x \sqsupset^h N_y$, iff :

1. $\forall (v_x, v_y) \in N_x \times N_y, v_x >^h v_y$ and
2. $\exists v \in V \mid v_x >^h v >^h v_y$.

(h-levels inclusion) The h-inclusion is the transitive closure of \sqsupset^h . It is denoted \sqsubset^h .

(\mathcal{Pr} and \mathcal{Sc}) Now we want to structurally distinguish between the primary and the secondary hierarchical levels, the first being indexed on the primary resources, and the second on additional, editorial resources.

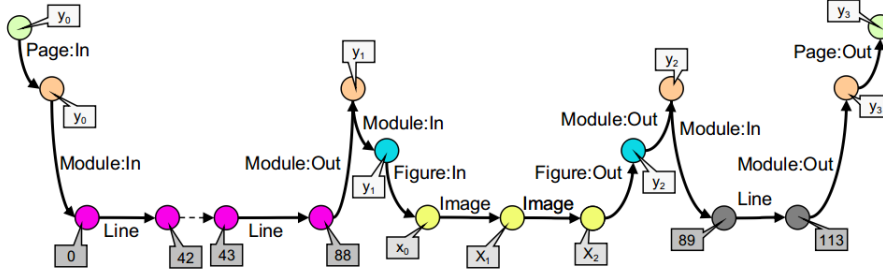
An h-level N is primary, denoted $N \in \mathcal{Pr}$ iff:

- N is among the top hierarchical level of the eAG, i.e. $\nexists N'$ so that $N \sqsubset^h N'$ and $root(G), leaf(G) \in N$.
- OR $\exists N' \in \mathcal{Pr}, N \sqsubset^h N'$ so that $\forall (v, v') \in N \times N'$, if $\exists e$ so that $v'[e]v \subseteq G \vee v[e]v' \subseteq G$, then:
 “:LinkTo” $\not\subset label(e) \Rightarrow$ (“:Att” $\not\subset label(e)$ and “:Com” $\not\subset label(e)$).

An h-level N is secondary, denoted $N \in \mathcal{Sc}$ iff:

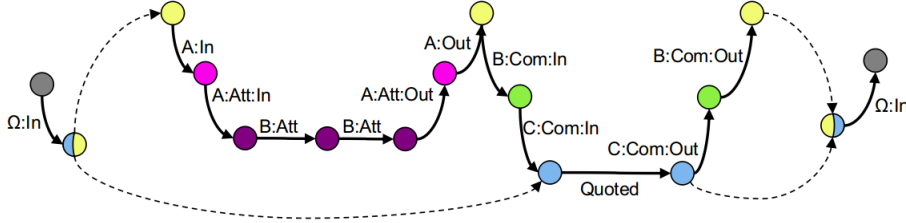
- N is among top hierarchical levels of a link, i.e. $\nexists N'$ so that $N \sqsubset^h N'$ while $\forall n \in N, in(n) > 0 \wedge out(n) > 0$
- OR $N \notin \mathcal{Pr}$ and $\forall N' \sqsubset^h N, \forall (v, v') \in N \times N'$, if $\exists e$ so that $v'[e]v \subseteq G \vee v[e]v' \subseteq G$, then:
 “:LinkTo” $\not\subset label(e) \Rightarrow$ (“:Att” $\subset label(e)$ or “:Com” $\subset label(e)$)

Illustration part 3-b. In the hierarchical annotation defined in Illustration part 3-a, the colours show the sets of nodes that define a h-level.



The blue h-level is directly included in the orange one, and (indirectly) included in the green one.

Example 5.1.3 Consider the following eAG:



The grey level is primary, because it is not inserted in any other one.
 The yellow and blue levels are primary, because they are included in the grey one by means of edges whose label does not contain **Att** or **Com**.
 The pink, purple and green levels are secondary.

Property 5.1.4 Be $G = (V, E)$. We enforce that for all N h-level of G , $N \in \mathcal{Pr} \cup \mathcal{Sc}$.

Definition 5.1.10 (Element) Be $G = (V, E)$ an eAG. Be $J \subset \mathbb{N}$, so that $\{H_j^L, j \in J, L \in \mathcal{L}\}$ is the set of rooted, connected and single-leaved subgraphs $H_j^L = (V_j^L, E_j^L) \subseteq G$ verifying, $\forall j$:

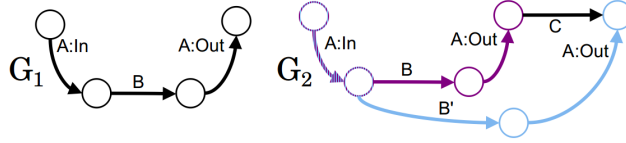
- a. $\exists e, e' \in E \mid \exists H_j^{L,int}$ rooted, single-leaved and connected, so that:
 1. $H_j^L = \text{root}(H_j) \mid e \mid \text{root}(H_j^{L,int}) \cup H_j^{L,int} \cup \text{leaf}(H_j^{L,int}) \mid e' \mid \text{leaf}(H_j^L)$
 2. AND $\text{label}(e) = L:\text{In}$ and $\text{label}(e') = L:\text{Out}$
 3. AND $\forall e''$ edge of $H_j^{L,int}$, $:\text{LinkTo} \notin \text{label}(e'')$
- b. $\forall X^L \subseteq G$ verifying condition a. and so that:
 $(\text{root}(X^{L,int}), \text{leaf}(X^{L,int})) = (\text{root}(H_j^{L,int}), \text{leaf}(H_j^{L,int}))$,
 then $X^L \subseteq H_j$.

The elements of G are the items of the set made out of the union of the sets of subgraphs U and $\{H_k^L, k \in K, L \in \mathcal{L}\}$ defined by:

- $U = \{v[e]v' \subseteq G\}$;
- $K \subseteq J$ so that $\forall j \in J, \exists k, l \in K \mid \text{root}(H_j^{L,int}) = \text{root}(H_k^{L,int}) \wedge \text{leaf}(H_j^{L,int}) = \text{leaf}(H_l^{L,int})$, and so that $\forall k, l \in K, k \neq l, H_k^{L,int} \not\subseteq H_l^{L,int} \wedge H_l^{L,int} \not\subseteq H_k^{L,int} \Rightarrow H_k^{L,int} \cap H_l^{L,int} = \emptyset$.

A well-formed eAG must finally conform to the following condition: $\forall v \in V$ so that there is $j \in J, L \in \mathcal{L} \mid v \in V_j^L$, then there is an element H_k^L so that $v \in V_k^L$.

Example 5.1.4 Consider the two following graphs G_1 and G_2 , both compliant with Definition 5.1.6-9 and Properties 5.1.2-4:

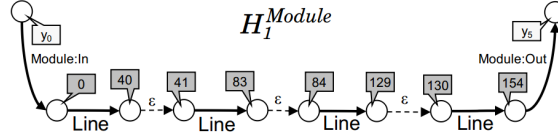


The first graph, G_1 , conforms to all the conditions in Definition 5.1.10, and itself, as a whole, is an element (relative to the label A), since it is the sole member of the most numerous set $\{H_j^A\}$ of graphs conforming to conditions a. to d. with $L = A$; it also contains a basic element (relative to the label B), made out of one edge and two nodes. It also conforms the well-formedness constraint expressed above. Actually, it is a canonical example of a hierarchical eAG.

The second graph, G_2 , is different in that two subgraphs H_1^A and H_2^A , respectively represented in purple and light blue, conform to the conditions a. to d. of the above definition, with the same $L = A$; still, the two graphs share their root. For that reason, according to the definition of the elements, only one among the two can be part of the elements' set (in which two graphs cannot share either root or leaf) ; yet if either the purple or the light blue graph was elected as an element, the well-formedness condition at the end of Definition 5.1.10 should not be satisfied: indeed, if the purple graph were chosen, then the leaf of the blue one, while taking part in an edge labelled $A:Out$, would not be part of an element related to the label A (and conversely if the blue graph was to be chosen). This means that the graph G_2 is not a well-formed eAG to begin with.

Forbidding two elements, related to the same label but not included one in the other, to share any edge may seem an arbitrary limitation to expressiveness. Still, *Illustration part 4* below shows that Definition 5.1.10 sets patterns that *look like* elements, but are not, apart; Paragraph 5.1.2.5 proves that the above restriction is *minimal* for the expression of multitrees in eAGs.

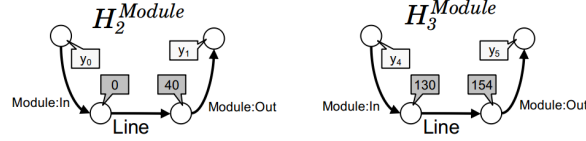
Illustration part 4. The following subgraph H_1^{Module} of the eAG represented on Figure 5.3 verifies the properties a. and b. listed above:



Still, it is not an element of the aforementioned eAG. Indeed, it is possible to find two other subgraphs H_2^{Module} and H_3^{Module} verifying conditions a. and b. as well, and so that:

- $root(H_2^{Module}) = root(H_1^{Module})$, while there is no other H_k^{Module} verifying a. to d. so that $leaf(H_k^{Module}) = leaf(H_2^{Module})$, and
- $leaf(H_3^{Module}) = leaf(H_1^{Module})$, while there is no other H_k^{Module} verifying a. to d. so that $root(H_k^{Module}) = root(H_3^{Module})$.

Those subgraphs H_2^{Module} and H_3^{Module} are the following:



On the contrary, those two subgraphs are two elements of the eAG we consider here, since they are part of the subset of all $\{H_k^{Module}\}$ of all the subgraphs verifying conditions a. to d. for $L = Module$ that abide by the last condition of Definition 5.1.10.

Property 5.1.5 Be $G = ((V, E), ref, label)$ an eAG. The above imply that $\forall e \in E$, $label(e) \in \mathcal{L}_{In}$ (resp. \mathcal{L}_{Out}) is the first (resp. last) edge of at most one element. We enforce that it shall be the first (resp. last) edge of at least one element also.

Comment. This is a way to ensure that any edge suffixed :In shall go with its corresponding :In edge, and that they define an element together.

Property 5.1.6 We impose the following properties regarding the content of each kind of elements, as defined in table 5.2:

1. An attribute element contains only attribute elements.
2. A Comment element may contain comments, attributes and elementary spanning elements.
3. An elementary spanning element may contain comments, attributes and elementary spanning elements.
4. Be L a link element. Then its root and leaf are nodes that belong to the attributes of some other element(s).

Comment. This property is consistent with the notion of covering chronologies (see Property 5.1.8 below).

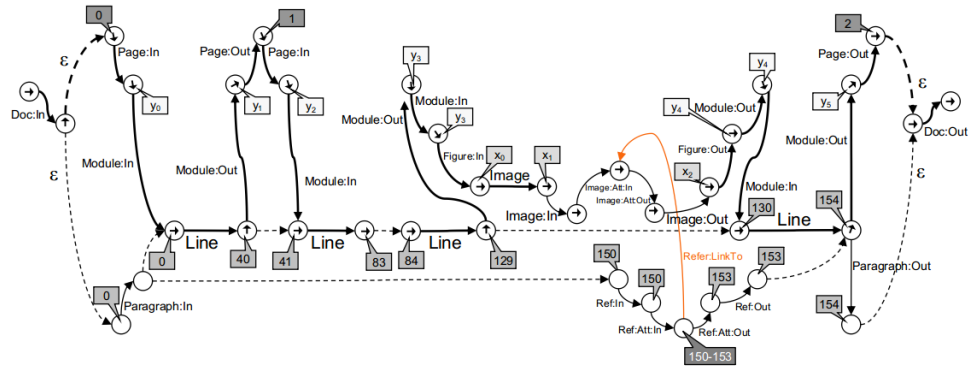


Figure 5.3: An eAG representing Figure 1 and a way to browse through it (arrows). Grey edges are for reading assistance (they span over the paths defining a structured element). Speech balloons show reference values (shades differentiate between chronometers), from a chronology extending $\langle T, \leq \rangle$ (cf. *Illustration, part 1*) in order to detail the content of Page one and a Ref (“[a]” in the text) between characters 150 and 153.

Property 5.1.7 Be $A \neq B$ two elements. We enforce that:

$$(\text{leaf}(A), \text{root}(A)) >_b^h \text{root}(B) \Leftrightarrow (\text{leaf}(A), \text{root}(A)) >_b^h \text{leaf}(B).$$

Comment. It means that two elements whose roots and leaves are either on the same h-level or on h-levels included one into the other either are *consecutive* (directly or not) or *include* one another. Paths connecting the root of an eAG to its node and made only out of consecutive and inclusive elements will be referred to as “linear annotation paths”. An eAG contains several such paths which, individually, represent a given annotation paradigm *à la* XML, since they can be modelled as ordered trees of elements. However, in an eAG, some elements can very well appear simultaneously on several such paths (c.f. *Illustration, part 4*). This means element hierarchies share items: an eAG can be modelled by no less than a multitree. Eventually, because edges whose label contains “:LinkTo” are unrestricted, they can connect any nodes together, which may result in a cyclic graph.

Illustration part 5. (Linear annotation paths) Figure 5.3 shows an eAG representing the document illustrated in Figure 5.2. It contains three competing linear annotation paths. The arrowed path provides a layout-oriented *Page* description, fragmented into *Modules*, *Lines*, *Figures* and *Images*. Another identifies a *Ref* inside the text of the *Paragraph*. The last path splits *Paragraphs* into *Lines*. *Lines* are *shared elements* with the first path; they are also *the only* shared elements. For instance, the *Paragraph* does not include the *Figure* element, since there is no h-inclusion between the h-levels where the roots and leaves of the two elements appear.

(Structured element, Link) The *Ref* element containing another element, which is its attribute element. *Ref* annotates the string “[a]” from the text. Graph-wise, it is

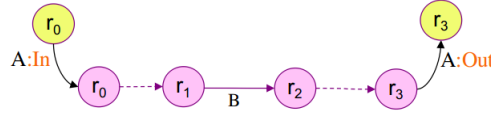
a structured element, since it contains more than one edge. It also points towards the second Image on another annotation path by means of an edge suffixed `:LinkTo`, originating in its attribute element.

Property 5.1.8 (Covering chronologies) We enforce that:

1. Be an h-level N . $\exists \langle T, \leq \rangle$ a chronology, $\exists ! c \in \mathcal{C}$ so that $\forall v \in N, \text{ref}(v) \in T \cap \mathbb{T}_c$. This defines a sub-chronology $\langle T_N, \leq_N \rangle$ so that $T_N = T \cap \mathbb{T}_c$ and $\leq_N = \leq_c$.
2. Be $(N, N') \in \mathcal{Pr}^2$. $N' \sqsubset N \Rightarrow \langle T_{N'}, \leq_{N'} \rangle \subseteq \langle T_N, \leq_N \rangle$.
- 2bis. Be $(N, N') \in \mathcal{Sc}^2$. $N' \sqsubset N \Rightarrow \langle T_{N'}, \leq_{N'} \rangle \subseteq \langle T_N, \leq_N \rangle$.
3. Be $v \lfloor_e \rfloor v'$ so that $\exists (N, N') \in \mathcal{Pr}^2$ (resp. \mathcal{Sc}^2) so that $(v, v') \in N \times N'$ and `:LinkTo` $\notin \text{label}(e)$. Point 1. or 2. (depending on $N = N'$ or $N' \neq N$) ensure that there is a chronology $\langle T, \leq \rangle$ so that $\text{ref}(v), \text{ref}(v') \in T$. Then $\text{ref}(v) \leq \text{ref}(v')$.

Comment. The above property implies that the reference values of the nodes of two consecutive h-levels of the same kind (either both \mathcal{Pr} or both \mathcal{Sc}) have to be comparable (in the sense that they can be assembled into a common chronology), so that the structural and the referential order between those nodes match. This is the condition for hierarchical annotation to be correct:

Let us consider the following eAG:



It contains two h-levels, the pink one being included in the green one. The annotation only makes sense if the span of the content annotated by the element B (ranging from r_1 to r_2) is encompassed by the span of the element A (ranging from r_0 to r_3). If so, then $r_0 \leq r_1 \leq r_2 \leq r_3$, which means that the sequence of reference values $[r_0; r_1; r_2; r_3]$ does constitute a chronology. Conversely, would the reference values not fit into a chronology in their order of appearance along the graph (e.g. if $r_1 < r_0$), the annotation would not make sense (B would be declared as included in an element that does not span over B).

Interestingly, the chronological requirement expressed in the above definition does not apply in two interesting cases: when the consecutive h-levels are not of the same kind (one being \mathcal{Pr} and the other one \mathcal{Sc}) or when no h-relation is defined between two nodes connected by an edge (which happens for links only – see Property 5.1.3.b. page 81).

However, because the above restriction only applies to linear hierarchical paths that, by definition, contain no `:LinkTo` edge, it is possible to define a linear path whose leaf has a smaller reference value than their root – those are links.

Regarding the situation in which $N' \sqsubset^h N$ with $N \in \mathcal{Pr}$ and $N' \in \mathcal{Sc}$: It is just normal that the reference values of the attributes of an element shall not be compared to the reference values of the nodes of the element itself, since the two do not index the same content at all.

Regarding the situation in which $N' \sqsubset^h N$ with $N \in \mathcal{Sc}$ and $N' \in \mathcal{Pr}$: due to Property 5.1.6, it may only happen in case of some Elementary spanning elements

being included in a comment element. This is what we call a **quote**, which is a special structural pattern that enables the editor to make explicit reference to some annotated content from the inside of a comment. An example of quote can be found in the eAG represented in Example 5.1.3 page 82. Here again, in order not to restrict the position of the quoted element in the annotation based on the position of the quoting comment, it is clear that we shall not require the reference values of the nodes of the quoting and quoted element to be neither comparable nor to be chronologically ordered. Importantly, because of the lack of this requirement, quotes may very well result in cycles in the graph – which is a first example why eAG graph structure is cyclic.

Eventually, because by definition, there is no h-relation between the nodes involved in edges whose label contains `:LinkTo`, the reference value of the root and the edge of a link element are unrestricted. This enables to link any pair of elements. Links also can result in cycles in the eAG (e.g. the link goes from the second of two consecutive elements to the first).

To conclude with, we provide the definition of two important features of eAG, that enable to express multilayer, overlapping annotation: Accidental nesting (or co-occurrence) and overlap:

Definition 5.1.11 (Accidental nesting) Be an eAG G and A, B two elements. B is accidentally nested in A iff there are:

- two linear annotation paths $P_1 = (V_1, E_1), P_2 = (V_2, E_2)$, their covering chronologies $\langle T_1, \leq_1 \rangle, \langle T_2, \leq_2 \rangle$ and $N_A \subseteq V_1, N_B \subseteq V_2$ the h-levels (in P_1 and P_2 resp.) so that $root(A), leaf(A) \in N_A$ and $root(B), leaf(B) \in N_B$, and
- $c \in \mathcal{C}, \exists N, N'; N \subseteq^h N_A, N' \supseteq^h N_B$ verifying $N \subseteq V_1, N' \subseteq V_2$, and $\exists (v_\chi, v_\phi) \in N^2, (v_x, v_y) \in N'^2$, so that:

1. $ref(v_\chi), ref(v_\phi), ref(v_x), ref(v_y) \in \mathbb{T}_c$
2. $ref(root(A)) \leq_1 ref(v_\chi), ref(v_\phi) \leq_1 ref(leaf(A))$
3. $ref(v_x) \leq_2 ref(root(B)), ref(leaf(B)) \leq_2 ref(v_y)$
4. $ref(v_\chi) <_c ref(v_x) <_c ref(v_y) <_c ref(v_\phi)$.

(Overlap) A and B overlap (with A first) iff the above paths, h-levels, chronometer and nodes exist and verify:

1. $ref(v_\chi), ref(v_\phi), ref(v_x), ref(v_y) \in \mathbb{T}_c$
2. $ref(root(A)) \leq_1 ref(v_\chi) \leq_c ref(v_x) \leq_2 ref(root(B))$
3. $ref(leaf(A)) \leq_1 ref(v_\phi) \leq_c ref(v_y) \leq_2 ref(leaf(B))$.

Example 5.1.5 In Figure 5.3, since the elements *Figure* and *Paragraph* are not on the same linear annotation paths, they cannot *include* one another. However, *Figure* is *accidentally nested* inside the *Paragraph*, since the *Paragraph* ranges from character 0 to character 154, values that compare with y_3 and y_4 on the covering chronology for the arrowed path.

Illustration part 6. (Linear annotation paths) One can check that it is possible to extend the composite chronology $\langle T, \leq \rangle$ defined in *Illustration, part 2* to cover the whole arrowed path in Figure 5.3, so that for any node v preceding a node v' along

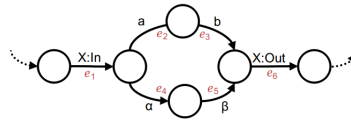
this path, $ref(v) \leq ref(v')$.

(Inter-chronometers comparisons) Cross-chronometer assessments can be made on an eAG. First example, because in $\langle T, \leq \rangle$, $x_2 < 130$, we know that from a (top-down) layout point of view, the second image precedes the last Line. (Cross-linear paths comparison) Cross-linear annotation path assessments can also be made, thanks to the notions of accidental nesting and overlap. E.g. *Ref* is accidentally nested in the last *Module*, because this *Module* h-cludes the *Line* delimited by characters 130 and 154, while *Ref* ranges from character 150 to 153. Then, it is possible to assess that the *Ref* shall be located further than the last *Image*, from a descending layout point of view. The edge labelled *Refer:LinkTo* (which is a link) does not respect the inferred reference order, which is not contradictory with the eAG model.

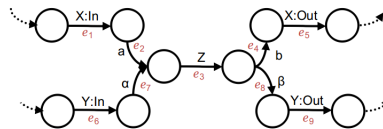
5.1.2.5 Elements in an eAG: Precisions

Definition 5.1.10 assesses that two elements related to the same label and not included one into the other will not share any edge or any internal node, that is, any node but their respective root or leaf. Let us call this property *Limitation E*. We briefly circumscribe the structural patterns such a limitation allows and forbids on eAGs. Then, we move on to define what a multitree may look like in eAGs. Eventually, we justify why Limitation E is minimal to allow a proper representation of multitrees in eAG.

Let us start by illustrating some canonical elements as defined by the Definition 5.1.10. First, an element may contain parallel annotation paths:

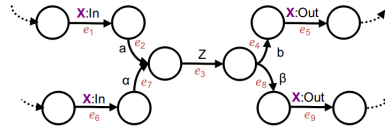


Second, elements related to different labels may intersect:



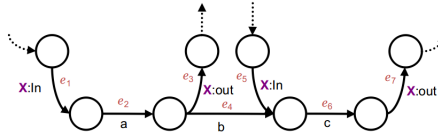
Those two patterns evidence that an eAG can represent overlapping elements (first example, elements a and β possibly, depending on the value of their respective right and left references) inside another element (related to label X , here), as well as multitrees (second example), that is, hierarchies of elements sharing subtrees (e.g. Z).

Still, not all multitrees can be represented. For instance, if one wanted to represent an element Z included, on the one hand, in an element X , along with an element a and an element b and, on the other hand, in *another* element X but this time along with two elements α and β , she could be tempted to draw the following graph:



Let us call the pattern represented here an X-pattern. The problem raised by this graph is that it is impossible to determine, by a structural analysis, which of the edges labelled on \mathcal{L}_{In} goes along with which edge labelled on \mathcal{L}_{Out} ¹¹. Considering each path bounded by a pair of edges labelled on \mathcal{L}_{In} and \mathcal{L}_{Out} respectively to define an element would contradict the multitree structure at stake, which does not specify the possibility to have a , Z and β , for example, included in the same element X . Hence the need to forbid such a situation.

Another situation that we do not want to forbid, but to disambiguate, is the one represented in Illustration part 4, page 83, which illustrates a situation typical of multitrees. In that case, the element *Line* is a leaf, shared by two hierarchies: one whose inclusive order of elements is *Page*>*Module*>*Line*, the other one *Paragraph*>*Line*. What happens is that the *Line* elements, that are included in separate *Module* elements, are connected by ϵ edges in the context of the *Paragraph* element; however, the fact that the first *Line* element follows a *Module* : In label, and the last *Line* element precedes a *Module* : Out label, yields a subgraph that mimics an element without being one. This situation can be synthetised as follows, and referred to as W-pattern:



Also, the question that arises in the light of the last two examples is: not any two edges labelled $X : In$ and $X : Out$, linked by a connected graph made out of parallel paths (called in-out pairs hereafter), define an element. Some conformations are illegal, and in others, we have to pick out the element-defining pairs.

From the analysis of the X-patterns, one can conclude that it must be illegal to have four in-out pairs (e_i^1, e_o^1) , (e_i^2, e_o^2) , (e_i^3, e_o^3) , (e_i^4, e_o^4) so that $e_i^1 = e_i^2 \neq e_i^3 = e_i^4$ and $e_o^1 = e_o^3 \neq e_o^2 = e_o^4$. Let us call this constraint *Limitation X*.

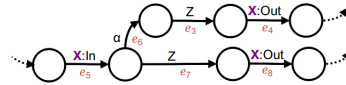
From the analysis of the W-patterns, we understand that among the set of in-out pairs, if there are three pairs (e_i^1, e_o^1) , (e_i^2, e_o^2) , (e_i^3, e_o^3) so that $e_i^1 = e_i^2$ and $e_o^1 = e_o^3$, and so that there is no other pair in which e_i^3 appears in first position and e_o^2 in second, then it means that the element-defining pairs are (e_i^2, e_o^2) , (e_i^3, e_o^3) . Let us call this criterion *Criterion W*. Indeed, the belonging of any edge to an in-out pair implies that, provided the graph is a well-formed eAG¹², the edge is part of an element (so that the last property in Definition 5.1.10 be verifiable); the exclusive belonging of an edge to an in-out pair implies that the pair does define an element.

¹¹Going back to the schema would not help. A schema defining such a multitree as described above would contain the same ambiguity as pointed here.

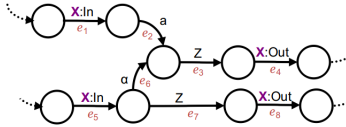
¹²By contraposition, a graph for which the selection of element-defining pairs shall leave edges that appeared in an initial pair out of any element is not well-formed.

At this point of our reasoning, one might notice that the above exclusiveness criterion of edges in the element-defining pairs is not equivalent to Limitation E. In particular, it does not forbid the pattern discussed in Example 5.1.4, that we might call the Y-pattern, in which two eligible subgraphs share either their **In** or their **Out** edge, but keep the other one exclusive. The reason why this pattern is forbidden follows:

Suppose that Y-patterns are allowed. Then, if we simply stick to Limitation X and Criterion W, in the following graph, one shall see two elements, one spanning from e_5 to e_4 , and the other one from e_5 to e_8 , the two being distinguished by the exclusive aspect of their **Out** edge (hypothesizing Criterion W to be sufficient):



Unlike the X pattern, the Y pattern does not trigger a combinatory correspondence of **In** and **Out** edges connected by a connected union of paths, and as such seems *meaningful*. It also abides by the Criterion W. The problem is that the YY-pattern below, which does not trigger a combinatory correspondence of **In** and **Out** edges connected by a connected union of paths either¹³, is rejected by the Criterion W:



Actually, since the YY-pattern makes no less sense compared to the Y-pattern, for each Y-pattern inside the YY-pattern can be interpreted independently, it seems that the Y-pattern is just a particular case of a pattern that cannot be legal, given we cannot do without Criterion W which disambiguates W-patterns that are typical of multitrees, and that we want eAGs to express multitrees.

Also, in order not to over-particularize Y-patterns, we decided to found the definition of elements on a stricter criterion than the Criterion W backed with Limitation X. We say that an element related to label X is defined by two exclusive in and out edges, that appear in no other X element, and that all edge labelled either in \mathcal{L}_{In} or in \mathcal{L}_{Out} must belong to an element. This is Limitation E. Note that it implies Limitation X, forbids both Y-patterns and YY-patterns, in a consistent approach, and enables the expression of multitrees, apart from those described previously.

5.1.3 Conclusion

To sum up, an extended Annotation Graph is a connected, directed and labelled graph whose nodes bear references values. An eAG is made out of smaller, structured subgraphs we call elements. Those can be of different kinds: elementary spanning elements, that play a similar role as elements in XML, but also attributes, that permit

¹³Indeed, turning a Y-pattern into a YY-pattern can be done by inserting *one* h-path, and this insertion adds only *one more pair* to the set of in-out pairs discussed on.

to qualify other elements further, links and quotes, that permit the insertion of critical information (notes and comments).

Structurally speaking, an eAG possesses one root and one leaf, connected together by paths that, individually, and provided they do not contain links, denote a hierarchical annotation – we call them *linear annotation paths*, in the sense that a linear annotation path is composed out of elements that either follow or include one another. Informally, an eAG is composed of several linear annotation paths sharing items and connected together by `:LinkTo` edges. The resulting graph is cyclic and permits the expression of multilayer annotation.

Now we want to define a schema model for eAG. Schemas are a means to define the allowed elements/attributes and their mutual relationships (consecutiveness, inclusion, existence of `:LinkTo` connexions) for the matching eAGs. Since elements, attributes and relationships have a homogeneous edge-based representation, eAG validation needs be no more than a *graph description* formalism, which simulation is [37]. Roughly speaking, a SeAG shall be a graph that simulates the eAG – see below.

Hence, in the next paragraph, we define SeAG, a simulation-based schema model for eAG. It relies on a notion of simulation we derive from the classic simulation relation: node-typed simulation. This notion rests upon the idea that the nodes of an eAG may be typed, that is, may be added a certain value, that implements the correspondence between the nodes of the eAG and the nodes of the schema. We also need to be able to make reference to nodes and edges individually, by means of an identifier.

Definition 5.1.12 (node types) Be \mathcal{T} and \mathcal{I} two infinite sets of values. We call $t \in \mathcal{T}$ a node type, $i \in \mathcal{I}$ an identifier.

Definition 5.1.13 (eAG node types, edge and node identifiers.)

Be $G = ((V, E), ref, label)$ an eAG. Then we define the function $type : V \rightarrow \mathcal{T}$, that associates a type value to each node of G . We also define the injective function $id : V \cup E \rightarrow \mathcal{I}$, that associates an identifier to each node and edge of G .

Nota. Because $type$ is a function, then it means (in particular) that a node has one and only one type. The same applies for id .

5.2 Schema Model

As we have seen, eAG belong to a very open family of graphs, namely, cyclic graphs. Yet an eAG is semantically structured into “linear annotation paths”, that are connected together by links. Those paths start at the root of the eAG, that is unique, and go until the leaf of the eAG, also unique. Thus, eAGs are “mostly linear”, and there is a natural pair of entry and exit nodes between which annotations read, as a sequence of labels. This reminds very much, actually, finite-state machines or automaton.

eAG validation can be enlightened by exploring further the analogy between eAG and automaton. We will start this presentation in that direction, by defining the notion of language of annotation. We will then show that validating a cyclic graph eAG annotation can be interpreted as restricting its language of annotation. A schema may

then be a graph, defining a (schema) language of annotation L_S and a valid instance shall then be a graph whose language of annotation L_I is included in L_S .

Second, we will see that this general principle can be operated by using simulation as a validation mechanism. Third, we compare simulation, as a validation mechanism, to tree-automaton-based simulation as implemented in relax-NG and show that, from a structural point of view, the two mechanisms are comparable for the validation of hierarchical annotations – while simulation extends to non-hierarchical annotations easily. Eventually, we discuss certain characteristics of SeAG schemas, including ambiguity and redundancy. This discussion will lead to the definition of a notion of equality for SeAG.

5.2.1 Finite-State Machine Analogy

5.2.1.1 The Notion of “Language of Annotation”

Extended Annotation Graphs are cyclic, labelled, directed and connected graphs, possessing one root and one leaf, so that each node of the graph is attainable from the root by a directed path. An eAG is also finite (its sets of nodes and edges are not infinite). If we set apart the fact that nodes bear a reference value, the eAG model matches the Finite-State Machine [134, 128] quite well.

Definition 5.2.1 : Finite-State machine. A Finite-State Machine (FSM) \mathcal{A} is a tuple (Et, Al, Tr, In, Fn) so that :

- Et is a finite set of states;
- Al is a finite set of labels ; it is also called the alphabet onto which the ;
- Tr is a relation on $Et \times Al \times Et$ called the *transition relation* ;
- $In \subset Et$ is the set of initial states of \mathcal{A} ;
- $Fn \subset Et$ is the set of final states of \mathcal{A} .

Definition 5.2.2 : set of FSM built on the alphabet Al . The set of FSM whose labels belong to Al will be denoted \mathcal{A}_{Al} .

Definition 5.2.3 : how it works. Be $\mathcal{A} \in \mathcal{A}_{Al}$. In intuitive words, a word $u \in Al^*$ is “accepted by \mathcal{A} ” iff there is a sequence of transitions allowed by \mathcal{A} from an initial state to a final state so that the concatenation of the labels from that sequence of transitions spells u .

Formally, the operation of $\mathcal{A} = (Et, Al, Tr, In, Fn)$ is defined as follows:

- A FSM configuration (S, u) is characterized by a set of states $S \subset Et$ and a string $u \in Ab^*$, that is the end of the input word to be analysed.
- The configuration (S', u') can be derived in one step from the configuration (S, u) (denoted $(S, u) \Rightarrow (S', u')$) iff $u = a.u'$ and $\forall v \in S, \exists v' \in S' \mid (v, a, v') \in Tr$.

- The configuration (S'', u'') can be derived in one step from the configuration (S, u) (denoted $(S, u) \xrightarrow{*} (S'', u'')$) iff (S'', u'') can be derived from (S, u) by a finite sequence of 1-step derivations.
- A word w is accepted by \mathcal{A} iff $(S_0, w) \xrightarrow{*} (S_f, \epsilon)$, where $S_0 \subset In$ and $S_f \subset Fn$.
- The language $lang(\mathcal{A})$ of the FSM is the set of the words \mathcal{A} accepts.

Notation. Be a FSM $\mathcal{A} = (Et, Al, Tr, In, Fn)$. For a given configuration (S, u) of \mathcal{A} , the FSM will be said to be in the state v iff $v \in S$. This situation will be denoted $\langle v \rangle$.

Definition 5.2.4 : ϵ transition in a FSM. There is a particular transition, classically denoted ϵ , defined as follows [128]: Be $(v, \epsilon, v') \in Tr$. Then $\forall (S, u), \langle v \rangle \Rightarrow \langle v' \rangle$.

Analogy between eAG and FSM. An eAG $G = (V, E)$ can be seen as a special case of FSM: 1) $V \equiv Et$; 2) $\mathcal{L} \equiv Al$; 3) $E \equiv Tr$; 4) $\{root(G)\} \equiv In$; 5) $\{leaf(G)\} \equiv Fn$. *Preuve.* Only point 3) requires a justification. It comes from Property 5.1.2 page 79, that asserts that in an eAG, an edge e can be identified by the triple $(sut(e), label(e), end(e))$.

In short, an eAG can be seen as a FSM with additional values on its nodes: type, reference values, identifier... Thus, we can define the “language” of an annotation graph as follows:

Definition 5.2.5 Be an eAG $G = (V, E)$. A sequence of labels $seq \in \mathcal{L}^*$ belongs to the annotation language of G iff seq is accepted by the FSM over \mathcal{L} corresponding to G .

The annotation language of G will be denoted $\mathcal{L}(G)$.

5.2.1.2 Regular Expression-based Language Representation

The language of FSM can be represented in a synthetic manner by means of regular expressions [181]. The notion relies upon the concept of language L over an alphabet Σ , \mathcal{L} being a subset of Σ^* .

Definition 5.2.6 Be L and L' two languages over the same alphabet.

- **Concatenation:** $L \cdot L' = \{xy; x \in L \wedge y \in L'\}$
- **Union:** $L|L' = \{x \in L, y \in L'\}$
- **Kleene closure:**

$$\begin{aligned}
 - L^0 &= \{\epsilon\} \\
 - L^k &= L \cdot L^{k-1} \\
 - L^* &= \bigcup_{i \in \mathbb{N}} L^i
 \end{aligned}$$

- **+ operator:** with the above notation, $L^+ = \bigcup_{i>0} L^i$
- **Optionality:** $L? = L \cup \{\epsilon\}$

Among those operators, $*$, $+$ et $?$ have the highest priority, followed by \cdot and then $|$.

Definition 5.2.7 : Regular expressions and related languages Be Σ an alphabet. let us denote RE the set of regular expressions on Σ . RE and the function $\mathcal{L}_{RE} : RE \rightarrow 2^{\Sigma^*}$ that associate a language to a regular expression are defined as follows:

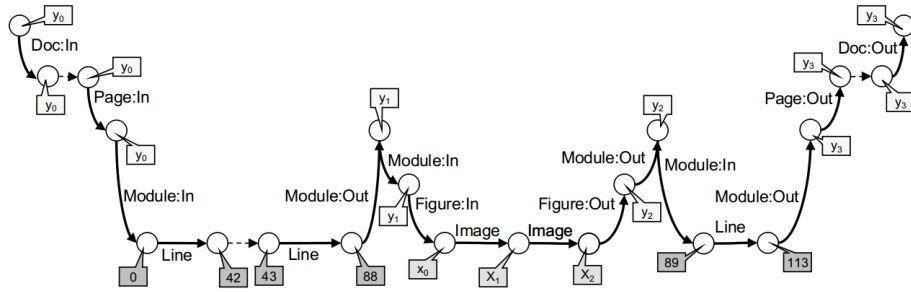
RE	\mathcal{L}_{RE}
<ul style="list-style-type: none"> • $\epsilon \in RE$ • $\emptyset \in RE$ • $\forall a \in \Sigma, a \in RE$ • $\forall E, F \in RE :$ <ul style="list-style-type: none"> – $E \cup F \in RE$ – $E \cdot F \in RE$ – $E^* \in RE$ – $E^+ \in RE$ – $E? \in RE$ 	<ul style="list-style-type: none"> • $\mathcal{L}_{RE}(\epsilon) = \{\epsilon\}$ • $\mathcal{L}_{RE}(\emptyset) = \emptyset$ • $\mathcal{L}_{RE}(a) = \{a\}$ • $\forall E, F \in RE :$ <ul style="list-style-type: none"> – $\mathcal{L}_{RE}(E \cup F) = \mathcal{L}_{RE}(E) \cup \mathcal{L}_{RE}(F)$ – $\mathcal{L}_{RE}(E \cdot F) = \mathcal{L}_{RE}(E) \cdot \mathcal{L}_{RE}(F)$ – $\mathcal{L}_{RE}(E^*) = \mathcal{L}_{RE}(E)^*$ – $\mathcal{L}_{RE}(E^+) = \mathcal{L}_{RE}(E)^+$ – $\mathcal{L}_{RE}(E?) = \mathcal{L}_{RE}(E) \cup \{\epsilon\}$

Property 5.2.1 RE is closed under the operations of union, concatenation and Kleen closure.

Definition 5.2.8 : Language inclusion. Be an alphabet Σ , $r, r' \in RE$. The language of r is included in the language of r' , denoted $\mathcal{L}(r) \subseteq \mathcal{L}(r')$ iff $\forall u \in \Sigma^*, u \in \mathcal{L}(r) \Rightarrow u \in \mathcal{L}(r')$.

5.2.1.3 Language of an eAG: Interpretation

To illustrate the notion of annotation language and how it may be connected to the notion of validation, let us take the example of an editor wanting to annotate the document provided on Figure 5.2 page 76, in terms of *Pages*, *Modules*, *Lines*, *Figures* and *Images*. A corresponding eAG is:



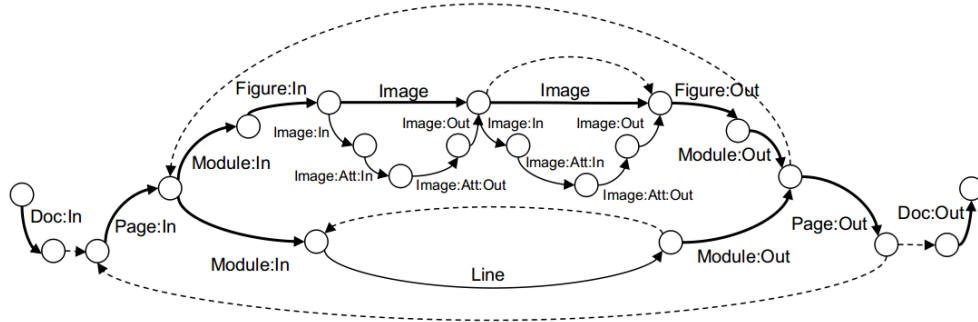
Since it is linear, the annotation language of the eAG is limited to one word u :

$$u = \text{Doc:In} \cdot \text{Page:In} \cdot \text{Module:In} \cdot \text{Line} \cdot \text{Line} \cdot \text{Module:Out} \cdot \text{Module:In} \cdot \text{Figure:In} \cdot \text{Image} \cdot \text{Image} \cdot \text{Figure:Out} \cdot \text{Module:Out} \cdot \text{Module:In} \cdot \text{Line} \cdot \text{Module:Out} \cdot \text{Page:Out} \cdot \text{Doc:Out}$$

This word from the language of annotation, which does describe the annotated document, can be seen as a particular instance of a general model for documents, assessing that a *Page* may contain one or more *Modules* of two kinds, depending on their content: the first kind contains either one or more *Lines* while the second contains one *Figure*, that being made out of one or two *Images*. This model can be represented by the following regular expression r_1 over \mathcal{L} :

$$r_1 = \text{Doc:In} \cdot (\text{Page:In} \{ \text{Module:In} \cdot \text{Line}^+ \cdot \text{Module:Out} \mid \text{Module:In} \cdot \text{Figure:In} \cdot \text{Image} \cdot (\text{Image})^? \cdot \text{Figure:Out} \cdot \text{Module:Out} \}^+ \text{Page:Out})^+ \text{Doc:In}$$

One possible automaton¹⁴ corresponding to this regular expression is the following:



As it happens, the language of the eAG is included in the language of annotation of this graph – as would be the language of an eAG annotating a *Page* with, for instance, four *Modules* each containing one *Line* and two *Figure*, etc.

And as it happens, this automaton also *simulates* the eAG above (see Paragraph 5.2.2.2).

Let us now shift from the hierarchical annotation of the document represented on Figure 5.2, and complement it with an additional description in terms of *Paragraphs* and internal *References*, that may point from the text towards an *Image*. As we have seen

¹⁴Indeed, several automaton may represent the same regular expression, as discussed in Paragraph 5.2.3.1.

previously, a corresponding eAG is given on Figure 5.3 page 85.

Now, the annotation language of the completed eAG contains more than one word: aside of u , there are also:

1. regarding the annotation in terms of *Paragraphs* and *Lines*:

$$v = \text{Doc:In} \cdot \text{Paragraph:In} \cdot \text{Line} \cdot \text{Line} \cdot \text{Line} \cdot \text{Paragraph:Out} \cdot \text{Doc:Out}$$

2. regarding the annotation in terms of *Paragraphs* and *Refs*:

$$w = \text{Doc:In} \cdot \text{Paragraph:In} \cdot \text{Ref:In} \cdot \text{Ref:Att:In} \cdot \text{Ref:Att:Out} \cdot \\ \text{Ref:Out} \cdot \text{Paragraph:Out} \cdot \text{Doc:Out}$$

3. regarding the existence of a link *Refer* between the *Ref* and the second *Image* in the *Figure*:

$$w = \text{Doc:In} \cdot \text{Paragraph:In} \cdot \text{Ref:In} \cdot \text{Ref:Att:In} \cdot \text{Refer:LinkTo} \cdot \text{Image:Att:Out} \cdot \\ \text{Image:Out} \cdot \text{Figure:Out} \cdot \text{Module:Out} \cdot \text{Module:In} \cdot \\ \text{Line} \cdot \text{Module:Out} \cdot \text{Page:Out} \cdot \text{Doc:Out}$$

Once again, the two first paths can be seen as an instance of a more general pattern:

1. The *Document* contains one or more *Paragraphs* each containing one or more *Lines*, which may be represented by the following regular expression:

$$r_2 = \text{Doc:In} \cdot (\text{Paragraph:In} \cdot \text{Line}^+ \cdot \text{Paragraph:Out} \cdot)^+ \cdot \text{Doc:Out}$$

2. Each *Paragraph* may also contain one or more *Refs*, that is an element containing a void attribute that will serve as the origin for a possible link:

$$r_3 = \text{Doc:In} \cdot [\text{Paragraph:In} \cdot (\text{Ref:In} \cdot \text{Ref:Att:In} \cdot \text{Ref:Att:Out} \cdot \\ \text{Ref:Out} \cdot)^* \cdot \text{Paragraph:Out}]^+ \cdot \text{Doc:Out}$$

Yet it has to be stated that the *Line* in the regular expressions r_1 and r_2 are the same, which is a way to say that a *Line* may belong both to a *Paragraph* and to a *Module*. A possible schema representing those two regular expressions taking into account the commonality of the *Line* element between r_1 and r_2 (and declaring the possibility of there being links between *Refs* and *Images*) is represented in Figure 5.4.

And as it happens, this SeAG, that schema that represents, in one move and harmoniously, the three regular expressions r_1 and r_2 and r_3 , the fact that that r_1 and r_2 share symbols, and the links, also *simulates* the eAG...

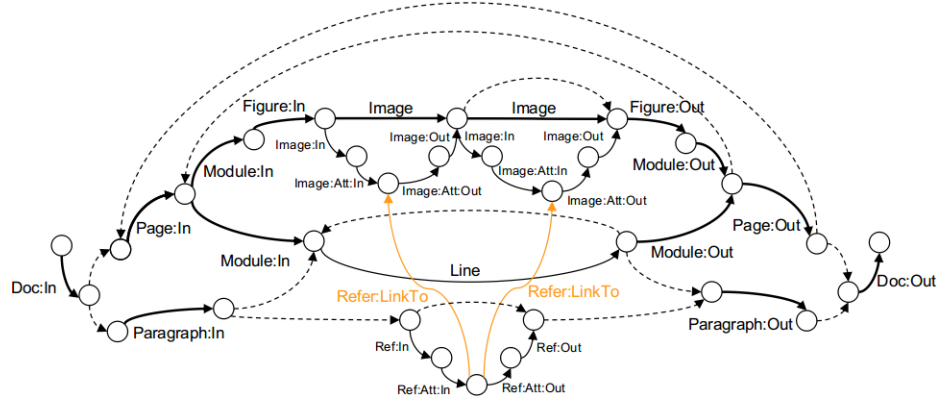


Figure 5.4: SeAG schema validating the eAG given in Figure 5.3.

5.2.2 The SeAG Model, Formally

The above qualitative introduction to eAG validation, provides – beside some key notions that will be helpful for the upcoming discussions (regular expressions, automaton, etc.) – with a philosophical statement, that can be summed up as follows: An eAG is made out of several interlaced linear annotation paths, connected together by means of links. The editorial information is thus conveyed by: 1) the sequence of labels along each hierarchical path; 2) the way those paths interlace; 3) the set of links that connect elements together; 4) the reference value of each node.

We claim that a Schema for eAG shall provide the user with the means to control the three first features, the reference value each node of the eAG shall bear being restricted by the eAG data model (and Property 5.1.8 in particular). For that purpose, as illustrated above, schemas can be graphs that roughly share the same syntax as eAG, whose annotation language covers the whole range of annotation that seem relevant to the editor. Simulation shall then be used as a validation mechanism.

5.2.2.1 SeAG Graph Model and Instantiation Function

Let us now define Schemas for eAG (SeAG) and their relation to eAG formally. Basically, SeAG edges share the same syntax as eAG edges; inclusion is defined a similar way, and so are elements. Still, SeAG nodes do not bear any reference value.

Nota. We remind the reader that an eAG G is a tuple $G = ((V, E), ref, label, type, id)$, where G is a rooted, single-leafed, directed, connected, labelled graph where $type : V \rightarrow \mathcal{T}$, is a function that associates a type value (an element of \mathcal{T}) to each node of G , and $id : V \cup E \rightarrow \mathcal{I}$ is an injective function that associates an identifier (an element of \mathcal{I}) to each node and edge of G .

Definition 5.2.9 An eAG schema S , denoted SeAG, is a tuple $((V_S, E_S), label, type, id, \delta)$, where (V_S, E_S) is a directed, connected, labelled graph with one root and one leaf only.

Its labels fall into Definition 5.1.2. It verifies Properties 5.1.1-7.

δ is the instantiation function, that is to be defined in Definition 5.2.2 below. Importantly, both functions $type : V_S \rightarrow \mathcal{T}$ and $id : V_S \cup E_S \rightarrow \mathcal{I}$ are injective.

Note that two nodes are not allowed to be connected by two edges with the same label (cf. Property 5.1.2).

Comment. The closeness of the eAG and the SeAG graph model and syntax, that is connected to the fact simulation will be defined as the validation mechanism for eAG, is also motivated by our problematics, that is to provide an annotation model in which schema amendments, that are to be operated by hand, shall translate into a transformation that shall translate, semi-automatically, the instance of the original schema into a possible instance of the amended schema. The closeness of the models of schemas and instances is designed to facilitate the derivation of that transformation on the instances from the amendment (i.e. transformation) on the schemas. See Part IV.

Nota. The most important difference between the eAG and the SeAG models, apart from the presence of the instantiation function δ , is the fact that two nodes of a schema cannot share the same type, while two nodes of an eAG can. The instantiation paradigm for eAG/SeAG couples rests upon that asymmetry:

Each node of a schema defines a type, characterized by a *type* value but also by its context, that is, by a series of labelled edges connected to other nodes of the schema. In a eAG that validates against a schema, each node of the eAG actually instantiates such a node type, that is, corresponds to one and only one node of the schema. Several nodes of the eAG may relate to the same node of the schema. The authorized edges between two nodes of the eAG are the labelled edges that exist between the nodes of the schema that correspond to them. This way, the eAG may only contain label sequences that can be read in the schema, and the way two linear annotation paths from the eAG interlace is also restricted by the schema.

This validation paradigm is embodied by the following δ function:

Definition 5.2.10 Be a SeAG $S = ((V_S, E_S), label, type, id, \delta)$. Be an eAG $G = ((V, E), ref, label, type, id)$. They form a schema-instance couple iff the instantiation function δ is defined on $E \cup V \rightarrow E_S \cup V_S$ so that:

- $\forall v \in V, \exists! v_S \in V_S \mid v_S = \delta(v)$
 $\forall e \in E, \exists! e_S \in E_S \mid e_S = \delta(e)$
- in particular, $root(S) = \delta(root(G))$ and $leaf(S) = \delta(leaf(G))$
- $\forall v, v_S \in V \times V_S, v_S = \delta(v) \Leftrightarrow type(v) = type(v_S)$
 $\forall e, e_S \in E \times E_S, e_S = \delta(e) \Leftrightarrow label(e) = label(e_S) \wedge sut(e_S) = \delta(sut(e)) \wedge$
 $end(e_S) = \delta(end(e))$
- $\forall (v [e] v') \subseteq G, (\delta(v) [\delta(e)] \delta(v')) \subseteq S$

Equivalent formulation. A SeAG $S = ((V_S, E_S), label, type, id, \delta)$ and an eAG $G = ((V, E), ref, label, type, id)$ form a couple iff $\forall v[e]v' \subseteq G, \exists!v_S[e_S]v'_S \subseteq S$ so that:

1. $type(v_S) = type(v)$
2. $type(v'_S) = type(v')$
3. in particular, $type(root(S)) = type(root(G))$ and $type(leaf(S)) = type(leaf(G))$
4. $label(e_S) = label(e)$

Definition 5.2.11: node and edge instantiation. Be S, G a schema-instance couple. A node $v \in V$ will be said to instantiate $v_S \in V_S$ iff $\delta_V(v) = v_S$, and an edge $e \in E$ will be said to instantiate $e_S \in E_S$ and $\delta_E(e) = e_S$.

5.2.2.2 Schema-Instance Relation: Node-typed Simulation

We have defined the validation paradigm that operates between schemas and instances. This relation can be formalized as a special kind of a well-studied relation, that is, simulation. Simulation in general was introduced by Milner [123] in the context of FSM. Intuitively, an automaton B simulates another automaton A if the output of B is the same as the output of A , for any input word. The notion was then diversified into specialized kinds of simulation for different purpose [149].

Simulation was first operated as a structure-descriptive mechanism for semistructured (S-S) data [171, 3]. S-S relied upon a cyclic, unordered, directed, edge-labelled graph model called the Object Exchange Model (OEM). In a S-S database, a Dataguide [86] or a Graph Schemas [37] is a graph inferred from the data, that simulates the data graph itself, so that it provides a structural description of the data that can be exploited for querying.

The classic definition of simulation is the following. In this definition, ϵ^* denotes a sequence of ϵ edges.

Definition 5.2.12 : Weak simulation. Be two directed, labelled graphs $A = (V_A, E_A)$ and $B = (V_B, E_B)$. A weak simulation of A by B is a relation $D \subseteq V_A \times V_B$ so that:

IF $(v_{A_1}, v_{B_1}) \in D \wedge (v_{A_1} \lfloor \epsilon^*.e_{A_1} \rfloor v_{A_2}) \subseteq A$, with $label(e_{A_1}) \neq \epsilon$,
 THEN $\exists v_{B_2} \in V_B \lfloor (v_{B_1} \lfloor \epsilon^*.e_{B_1} \rfloor v_{B_2}) \subseteq B \wedge label(e_{B_1}) = label(e_{A_1}) \wedge (v_{A_2}, v_{B_2}) \in D$.

For rooted graphs, it is required that the pair of roots belong to D .

Let us denote the fact there is a weak simulation of A by B : $B \xrightarrow{typ.} A$.

The above definition is clearly FSM-oriented, since it considers ϵ edges as special transitions that are consumed whatever the input letter. A more graph-oriented version of simulation was proposed in [149], where all the edges are considered equally, regardless of their label. This is called strong simulation, in the sense that the existence of a strong simulation between two graphs implies the existence of a weak simulation between those graphs, but not conversely.

Definition 5.2.13 : Strong simulation. Be two directed, labelled graphs $A = (V_A, E_A)$ and $B = (V_B, E_B)$. A strong simulation of A by B is a relation $D \subseteq V_A \times V_B$ so that:

IF $(v_{A_1}, v_{B_1}) \in D \wedge (v_{A_1} [e_{A_1}] v_{A_2}) \subseteq A$,

THEN $\exists v_{B_2} \in V_B | (v_{B_1} [e_{B_1}] v_{B_2}) \subseteq B \wedge label(e_{B_1}) = label(e_{A_1}) \wedge (v_{A_2}, v_{B_2}) \in D$. Let

us denote the fact there is a strong simulation of A by B : $B \xrightarrow{strong} A$.

Regardless of the kind of simulation, we have the following property.

Definition 5.2.14 : maximal simulation. Be two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Be $D \subseteq V_1 \times V_2$ a simulation of G_1 by G_2 .

This relation is "the maximal simulation of G_1 by G_2 " iff for all $D' \subseteq V_1 \times V_2$, simulation of G_1 by G_2 , $D' \subseteq D$.

Theorem 5.2.1 If there is a simulation of G_1 by G_2 , then there is a maximal simulation of G_1 by G_2 [38].

On top of this definition, we build the following notion of node-typed simulation, that takes into account the notion of node-types.

Definition 5.2.15 : node-typed simulation. Be two directed, labelled graphs $A = ((V_A, E_A), type)$ and $B = ((V_B, E_B), type)$ whose nodes are typed over the same type set \mathcal{T} . A node-typed simulation of A by B is a strong simulation D of A by B so that $\forall (v_A, v_B) \in D, type(v_A) = type(v_B)$.

Let us denote the fact there is a node-typed simulation of A by B : $B \xrightarrow{typ.} A$.

Property 5.2.2 Be two directed, labelled graphs $A = ((V_A, E_A), type)$ and $B = ((V_B, E_B), type)$ whose nodes are typed over two type set \mathcal{T}_A and \mathcal{T}_B , so that $B \xrightarrow{strong} A$. Then there is a retyping of the nodes of A so that $B \xrightarrow{typ.} A$.

Proof. Let us denote D the strong simulation of A by B . It suffices to define the retyping function $retype : \mathcal{T}_A \rightarrow \mathcal{T}_B$ so that $\forall (v_A, v_B) \in D, retype(type(v_A)) = type(v_B)$.

We now turn to the following result:

Theorem 5.2.2 Be $S \cdot I_S$ a schema-instance couple, and δ the corresponding instantiation function. This function defines a node-typed simulation I_S par S .

Proof. Be I_S, I_S a schema-instance couple.

We check that $\delta = (\delta_V, \delta_E)$ defines a node-typed simulation of de I_S by S :

Be $D = \{(v, v_S) | v \in V \wedge \delta_V(v) = v_S\}$.

Be $v, v' \in V, e \in E$ so that $(v, v_S = \delta(v)) \in D \wedge (v [e] v') \in I_S$.

Ten, by definition of δ , $\exists v'_S \in V_S, e_S \in E_S$ so that $\delta(v') = v'_S, \delta(e) = e_S$ and so that $(v_S [e_S] v'_S) \in S \wedge (v', v'_S) \in D$.

To conclude, we notice that :

- $type(v) = type(\delta(v))$, hence $type(v) = type(v_S)$ (idem, $type(v') = type(v'_S)$) ;
- $label(e) = label(\delta(e))$, hence $label(e) = label(e_S)$ ■.

Definition 5.2.7: validation. Be S a SeAG and G an eAG. The S validates G iff there is a retyping of G so that $S \xrightarrow{typ.} G$, so that the pair of the roots of S and G and the pair of their leaves belong to the maximal node-typed simulation.

Validation: use cases. The above definition covers two cases: when the retyping is necessary and when it is not. The two cases do not correspond to the same use case. First, an eAG is freely constructed by the editors and then, is confronted to a certain schema in order to check validation a posteriori. In this case, there is no particular reason why an eAG and a schema, that were defined independently, shall be defined so that their types belong to the same type set, or so that their types match, even if, structurally speaking, the eAG expresses an annotation that, modulo a retyping, S validates.

The second case may happen if, *provided a schema*, an eAG is constructed in order to match the schema. In this case, validation may not be done a posteriori, but the schema may be used, in the course of the eAG manufacturing, as a guide for annotating the data, just the way XML schemas can be used in an XML editor featuring content assist (e.g. depending on the context the editor is annotating, the editor may suggest the possible XML elements, according to the schema). We may talk about *on-the-fly validation*, to insist on the fact validation is done during the data manufacture, or validation by construction, if the manufacture process makes use of on-the-fly validation in a way that does not permit the expression of non-valid annotations. SeAG is designed so that both kinds of validation shall be possible. A complete discussion of this aspect is given in Paragraph 5.3.

The following result makes a connection between simulation-based validation and the notion of language of annotation introduced above.

Theorem 5.2.3 Be two rooted graphs A and B . Then $B \hookrightarrow A \Rightarrow \mathcal{L}(A) \subseteq \mathcal{L}(B)$.

Proof. Be $(A = (V_A, E_A), B = (V_B, E_B))$ a couple of graphs.

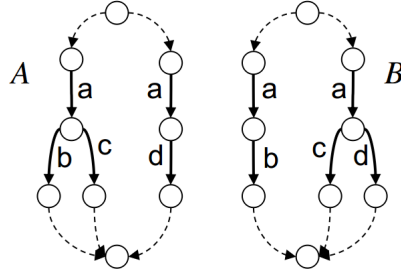
Be D the maximal simulation of A by B .

Be $\mathcal{L}(A)$ a word containing more than one letter (the case of the empty word is trivial). Let m_i refer to the i^{th} letter of m . By recurrence, let us demonstrate that $m \in lang(S') \Rightarrow m \in lang(S)$.

- $m \in lang(S') \Rightarrow \exists e'_0 \in E_{S'}, v'_0 \in V_{S'} \mid r(S')[\epsilon^*.e'_0]v'_0 \in S' \wedge label(e'_0) = m_0$.
Yet $S \hookrightarrow S' \Rightarrow (r(S'), r(S)) \in D$
 $\Rightarrow \exists e_0 \in E_S, v_0 \in V_S \mid r(S)[\epsilon^*.e_0]v_0 \in S$
 $\wedge label(e_0) = label(e'_0) = m_0$.
- Be $i \leq N - 1 \mid \exists \{e'_j \mid label(e'_j) = m_j; 0 \leq j < i\} \in E_{S'}^i$.
Yet $m \in lang(S') \Rightarrow \exists e'_i \in E_{S'}, v'_i \in V_{S'} \mid v'_{i-1}[\epsilon^*.e'_i]v'_i \in S' \wedge label(e'_i) = m_i$.
Yet $S \hookrightarrow S' \Rightarrow \exists e_i \in E_S, v_i \in V_S \mid v_{i-1}[\epsilon^*.e_i]v_i \in S$
 $\wedge label(e_i) = label(e'_i) = m_i$ ■.

Nota. Since $S \xrightarrow{typ.} I_S \Rightarrow S \hookrightarrow I_S$, then $S \xrightarrow{typ.} I_S \Rightarrow lang(I_S) \subseteq lang(S)$.

Comment. The converse is wrong. The following graphs have the same language but do not simulate one another:

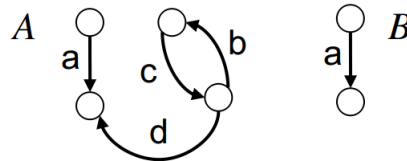


This means that simulation not only controls the annotation language of an instance, but also concerns the shape, the structure of the graph. Let us now investigate how.

eAG/SeAG graph model and simulation. The fact that $B \hookrightarrow A$ does not suffice, in general, to ensure that all the sequences of labels present in A be included in the set of sequences present in B . This results from the fact we consider rooted simulations for connected graphs.

Indeed, if we do not consider rooted graphs for which the roots must be related by simulation, it is easy to find counter examples. For instance, two small linear graphs whose label sequence reads $\mathbf{a}\cdot\mathbf{X}$ and $\mathbf{b}\cdot\mathbf{X}$ simulate one another, while their languages do not intersect.

We can also define a pair of rooted graphs A and B , so $B \hookrightarrow A$, so that their roots are related by simulation, but that lack the connectivity property (stating that all the nodes shall be reachable from the root of the graphs), so well so that $\mathcal{L}(A) \not\subseteq \mathcal{L}(B)$ – see below.



This highlights the compatibility of the eAG and SeAG graph model with simulation-based validation.

5.2.2.3 SeAG Expressive Power

Let us now illustrate, based on this set of definitions, how simulation can be used to express whole families of graphs with a restricted annotation language and structure, that is, can be used to model data.

First, we establish some that simulation is closed under some operations (union, concatenation, intersection, as defined below), that permit to design a schema in a modular approach, similar to the FSM construction from regular expressions.

Property 5.2.2.3.1: Closure under union. Be (S, I) and (S', I') two SeAG/eAG pairs. Then $S \cup S' \xrightarrow{typ.} I \cup I'$ and the maximal simulation of $I \cup I'$ by $S \cup S'$ is total.

Proof. Simulation is closed under union – see [144]. Since types are not affected by the union of two graphs, if D is the maximal simulation of I by S and D' the maximal simulation of I' by S' , then $D \cup D'$ is a node-typed simulation of $I \cup I'$ by $S \cup S'$. Thus, since D and D' are total, this means that there is a total simulation of $I \cup I'$ by $S \cup S'$.

Nota. It follows that a the graph obtained by connecting the two roots of $S \cup S'$ to a unique root by means of two ϵ edges, and the two leaves to a unique leaf by means of ϵ edges, also node-typed simulates the graph obtained by connecting together I and I' the same way. See Figure 5.5.a.

Property 5.2.2.3.2 Be S a graph simulating another graph I . Then $\forall G$ so that $S \subseteq G$, then there is a (potentially non-rooted) simulation of I by G .

Proof. It suffices to consider that given two graphs A and B so that B simulates A , adding an edge to B does not change the fact that B is simulated by A . It may however prevent there being a rooted simulation, that is, a simulation in which the roots of A and if they are rooted graphs, belong.

Property 5.2.2.3.3: Closure under concatenation. Be (S, I) and (S', I') two SeAG/eAG pairs. Be $S \cdot S'$ the graph obtained by connecting the leaf of S to the root of S' by means of one ϵ edge, and $I \cdot I'$ the graph obtained by connecting the leaf of I to the root of I' by means of one ϵ edge. Then $S \cup S' \xrightarrow{typ.} I \cup I'$ and the maximal simulation of $I \cup I'$ by $S \cup S'$ is total.

Proof. Let us denote v_S and $v_{S'}$ the nodes in $S \cdot S'$ that corresponds to $leaf(S)$ and $root(S')$ respectively; let us denote v_I and $v_{I'}$ the nodes in $I \cdot I'$ that corresponds to $leaf(I)$ and $root(I')$ respectively.

First, because of Property 5.2.2.3.2, $S \cdot S'$ simulates I , so well so that $(v_S, v_I) \in D$.

First, for the same reason, $S \cdot S'$ simulates I' , so well so that $(v_{S'}, v_{I'}) \in D$. We also know that there is an edge going out of v_S in $S \cdot S'$ is an ϵ edge leading to $v_{S'}$. Then, it is easy to check that $S \cdot S'$ simulates $I \cup v_I[\epsilon]v_{I'}$, so that $(v_{I'}, v - S')$ is a pair of the relation.

To conclude, since $I \cup v_I[\epsilon]v_{I'} \cup I' = I \cdot I'$. Since simulation is closed under union, then it follows that $S \cdot S'$ simulates $I \cdot I'$. The above demonstration tends to show that is D is the maximal simulation of I by S and D' the maximal simulation of I' by S' , then $D \cup D'$ is a node-typed simulation of $I \cdot I'$ by $S \cdot S'$.

Nota. It follows that a the graph obtained by connecting the leaf of S to its root by means of ϵ will simulate any concatenation of instances of S , but also any set of parallel concatenations of instances, according to Property 5.2.2.3.1. See Figure 5.5.b.

Discussion, part 1. Figure 5.5 represents the two core patterns that can be used in a SeAG, either to model a. the fact there can be locally two competing, or alternative, models of the corpus (e.g. a *Module* contains either *Lines* or a *Figure*) – which is

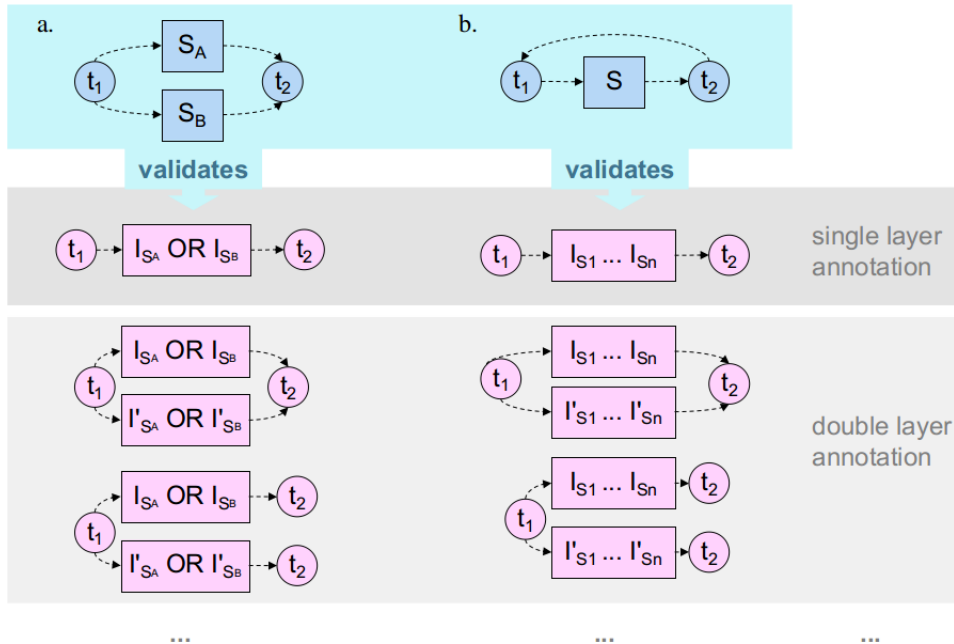
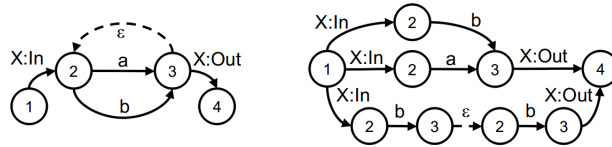


Figure 5.5: Two fundamental properties of simulation-based validation. a. Two SeAG S_A and S_B patterns in parallel may validate either any instance of one of the patterns, or any superposition of instances of one of the patterns. b. A SeAG pattern S made cyclic will validate any concatenation of any of its instances, provided the concatenation is well-formed, or any superposition of such concatenations.

represented by two parallel paths in the global schema – or b. to represent the fact that some elements, or sequence of elements, may occur several times in a row (e.g. a *Paragraph* contains one or more *Lines*). In other words, those properties provide with a natural SeAG representation for the regular-expression operators OR and *.

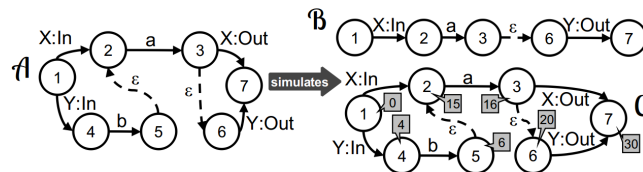
Example 5.2.1 Consider the schema A below, where the values in the nodes are their types. It is the Ott automaton [128] representing the regular expression $r = X:In(a|b)^*X:Out$. Consider the eAG B , whose label sequences along the linear annotation paths are words from the language of r .



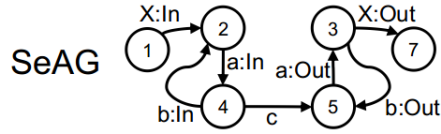
A and B form a schema-instance couple.

Discussion, part 2. Yet, the above automaton interpretation of SeAG only holds because of the well-formedness constraints for eAGs. For instance, the interpretation of the *cyclic* SeAG A in Example 5.2.2.3 as an Ott automaton, i.e. as a means to define an infinite set of *acyclic paths*, would not be correct without Property 5.1.8. Indeed, there is at least one cyclic graph that A simulates: A itself, but there is no way a graph structurally identical to A shall be an eAG. Consider A equipped with reference values on its nodes. Property 5.1.8, states that edges not suffixed `:LinkTo` go from nodes with a lower reference value to nodes with a higher one. Since the cyclic subgraph made out of the edges between the nodes typed 2 and 3 contains no `:LinkTo`, then whatever the reference values of its nodes, it cannot be part of an eAG. Conversely, the cycle in A will only be instantiated by acyclic paths (cf. graph B , Example 5.2.2.3) in the eAGs validated by A .

More generally, eAG data model and simulation-based validation make sense *together*. The following examples illustrate how the definition rules for eAG give sense to the SeAG formalism. First, an SeAG can express that two h-levels share elements: see graph A below. This SeAG does simulate the faulty graph B , where the inclusion semantics is lost (e.g. `X:Out` is missing), but since *for that reason*, regardless of its nodes references, B is not well-formed (see Property 5.1.5), it is not to be considered for validation. However, A *validates* the *well-formed* eAG C , which implements properly multitree annotation with shared items between h-levels.



Thanks to the same Property 5.1.5 in the eAG data model, an SeAG can contain recursive elements as well :



5.2.2.4 SeAG Expressive Power: Anaphoric Chains Validation

An interesting aspect of SeAG is that it is fit for validating two kinds of annotations that can be expressed in eAG, and that are hard to validate, namely, cyclic annotation and multilayer annotation.

Multilayer annotation SeAG schemas actually define two kinds of multilayering, as suggested by Figure 5.5.a. In this Figure, the pattern that is discussed presents two parallel halves, each containing an instantiable graph, that is, a graph that respects the SeAG model. Each half thus defines a possible annotation paradigm. By property of simulation, as shown in the same Figure, this ‘alternative’ pattern simulates:

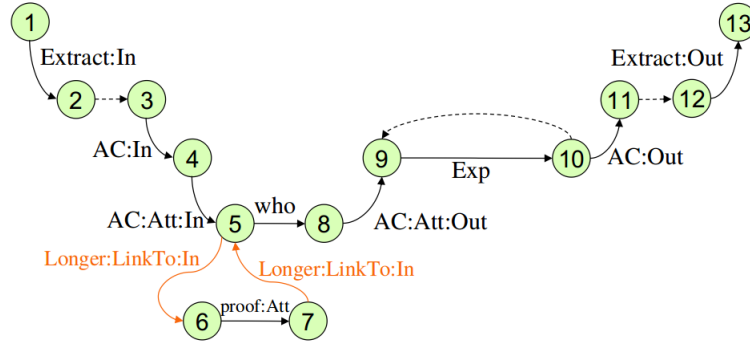
1. either a graph instantiating just one of the two paradigms;
2. or a two layered graph, whose layers are parallel graphs each instantiating each one of the paradigms;
3. or a two layered graph, whose layers are parallel graphs both instantiating the same paradigm;
4. or a multilayer graph, whose layers instantiate any of the two paradigms...

The item 1 above goes along with the interpretation of the schema pattern as expressing optionality.

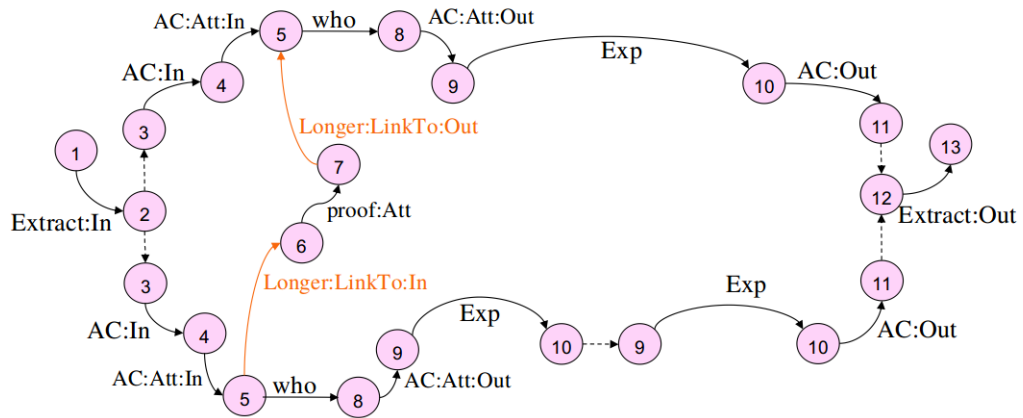
The item 2 means that when the schema contains two parallel, or alternative paths, those define two different ways to annotate the same content: hence, the schema will validate an annotation containing two layers, each instantiating one of the two paths of the schema. Let us call this kind of multilayering **Schema-based multilayering**. The item 3 really comes from a peculiar property of simulation, that can be condensed as follows: be a graph made out of one edge labelled *A*; then this graph will simulate any graph made out one or more edges labelled *A* in parallel (that is, so that two edges do not form a path). In the SeAG context, this property means that any path from the schema may be instantiated several times onto the same content, hence resulting in a particular kind of multilayering, in which the different layers do not instantiate distinct annotation paradigms, but all the same. Let us call this **Simulation-based multilayering**.

Schema-based multilayering certainly seems a natural notion: when the editors need to annotate the same content with different views (e.g. materially vs. linguistically), more than one paradigm needs to be defined for that content, and will be instantiated one on top of the other. Simulation-based multilayering, even if it may seem less familiar at first sight, is also a useful notion to validate self overlapping annotation, that is exemplified by the Anaphoric Chains annotation.

Let us indeed build a schema for Anaphoric Chains annotation. What we want to say is that *Anaphoric Chains* (*AC*) contain one or more *Singular Expressions* (*Exp*); that they are defined within the scope of an *Extract*; that we may be interested in reifying the relation ‘this *AC* contains more *Exp* than that *AC*’ by means of a structured link *Longer:LinkTo*, containing a *proof:Att* element. A corresponding schema is the following:



We remind the reader that the eAG we came up with for the representation of the anaphoric chains regarding the beating and the observer from *the Village of Ben Suc* extract presented in Paragraph 5.1.1 page 71 is the following – equipped with node types this time (reference values are omitted):



Indeed, this eAG instantiates the same hierarchical path from the schema twice, in order to locate the two anaphoric chains that overlap in this extract.

Links Validation of links comes from the fact that simulation is a relation defined for cyclic graphs. A link, in the eAG/SeAG syntax, is a special element whose name contains the suffix *:LinkTo*; apart from that, the element itself can be structured just like any other, that is, it may contain other elements, in order for the link to convey

more than the relation information, but possibly critical enlightening about that relation. The context of a link, in an eAG, is yet particular: contrary to all the other elements, a link does not take place in hierarchies of elements: instead, the root of a link is a node from the attributes of a source element, and its end is a node from the attributes of a target element.

SeAG schemas enable to control both of the above aspects of a link, namely: its structure and its context. See the example of SeAG for anaphoric chains above. The structure of the link `Longer:LinkTo` is controlled by the schema just as the structure of any other element: here, an instance of the link can be nothing but a sequence of three edges `Longer:LinkTo:In`, `proof:Att` and `Longer:LinkTo:Out`. The context of the link is also restricted by the schema: the root of any instance of a link will be a node whose type is 5, and will end on a node with the same type. Since the only nodes of type 5 are involved in defining the attribute element of AC, then it means that the link will connect two ACs only.

5.2.2.5 Simulation-based Validation: Caveats

Caveats [37, 3] point out several limitations to simulation-based graph description. First, for a given instance graph, several schemas are eligible, since simulation is transitive. This matters greatly when schemas are inferred from the data, but does not when they are predefined.

Second, and more importantly, simulation-based validation as defined by [37] does not enforce the presence of a label. This is true for simulation between two general graphs. Still, this caveat can be bypassed by specifying an appropriate data model. Consider the SeAG A and the graphs B_1 and B_2 below. Even though A simulates both B_1 and B_2 , it validates none: B_1 is not well-formed, and $type(leaf(B_2))$ is not equal to $type(leaf(A))$, which contradicts the validation definition.



Hence well-formedness and validation rules somehow enforce the presence of labels that simulation does not.

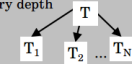
Last, simulation cannot prevent a node from having several outgoing edges. Said differently, as illustrated in Example 5.2.2.3, even when an SeAG contains one single (cyclic) path, there is no way to prevent the annotator to annotate the same content with several layers all instantiating the same path. Of course, this feature has positive aspects – see the previous paragraph and the notion of Simulation-based multilayering. But it means that a hierarchical SeAG will validate multitrees, not trees only.

5.2.2.6 Simulation-based Validation vs. Grammar-based Validation (for Trees)

Yet there is a connexion between simulation and grammar-based validations. An XML document is, syntactically speaking, a tree; a (RelaxNG) schema is a Tree automaton

Tree automaton TA

$S = \{\text{doc}\}$ [NB. S is a subset of N]
 $N = \{\text{doc, page, module, figure, image, line}\}$
 $T = \{\text{Doc, Page, Module, Figure, Image, Line}\}$
 $R = \begin{cases} \text{doc} \rightarrow \text{Doc}(\text{page}^+) \\ \text{page} \rightarrow \text{Page}(\text{module}^+) \\ \text{module} \rightarrow \text{Module}(\text{line}^+ \mid \text{figure}) \\ \text{line} \rightarrow \text{Line}() \\ \text{figure} \rightarrow \text{Figure}(\text{image} \cdot \text{image}?) \\ \text{image} \rightarrow \text{Image}() \end{cases}$

Interpretation :
 An interpretation is a function $I: T \rightarrow N$ so that :
 1. $I(\text{root}(\text{Tree}))$ is in S and
 2. for all maximal subtree of unitary depth

 there is $r = t \rightarrow T(RE)$ in R so that:
 - $I(T) = t$ and
 - $I(T_1) \cdot I(T_2) \dots \cdot I(T_N)$
 is in language of RE .

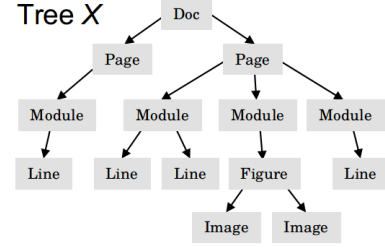


Figure 5.6: RelaxNG tree automaton-based XML validation mechanism.

[126], which validates the tree for which there is an “interpretation”, as defined in Figure 5.6. We know there is a way to translate trees into eAGs. For instance, the eAG representing the tree X from Figure 5.6 is the arrowed path in Figure 5.3. There is also a way to derive a SeAG from a Tree automaton. Consider an automaton $TA = (S, N, T, R)$, for instance the one from Figure 5.6. In TA , T is the set of terminals. N is the set of non-terminals, among which are start symbols (S). The elements of R are called production rules. A rule associates a non terminal to a terminal, representing a possible labelled node of a tree, and a regular expressions over N against which the sons of the node shall match.

For our derivation of an SeAG from a TA, we enforce that if there is a rule $r = x \rightarrow X(\text{reg}) \in R$ so that reg can be expressed as $\text{reg}_1|\text{reg}_2$, with no common prefix and suffix between the words in the languages of reg_1 and reg_2 , then r must be split into two rules $r_1 = x \rightarrow X(\text{reg}_1)$ and $r_2 = x \rightarrow X(\text{reg}_2)$. For instance, $\text{module} \rightarrow \text{Module}(\text{line}^+|\text{figure})$ shall be split into two rules $\text{module} \rightarrow \text{Module}(\text{line}^+)$ and $\text{module} \rightarrow \text{Module}(\text{figure})$. Then, the derivation of a SeAG S_{TA} from TA ¹⁵ defines as follows:

There is a partition of R into sets of rules sharing the same left-hand side. For any $n \in N$, let us call R_n one such subset of R . Then, every $n \in N$ may define what we call a unique Box, denoted $n\text{Box}$. The $n\text{Box}$ is a rooted graph that reflects the content of the set of rules in R_n . In the $n\text{Box}$, each $r_i = n \rightarrow T_i(\text{re}_i) \in R_n$ is represented by a root-to-node path. If $\text{re}_i = \emptyset$, then the path is a single edge labelled T_i . Else, since re_i is a regular expression over N , it can be represented by the Ott automaton made out of the Boxes corresponding to re_i , escorted by two edges labelled $T_i:\text{In}$ and $T_i:\text{Out}$. Figure 5.7 shows the $n\text{Boxes}$ for the automaton in Figure 5.6. By replacing iteratively, in a bottom-up approach, the Boxes contained one in the others, we get a labelled graph which is S_{TA} . One can check that, in the case of Figure 5.7, this yields the SeAG shown on Figure ??.

Here comes the interesting point: this example illustrates that, given a tree automaton TA and a tree X so that there is an interpretation of X against TA , the SeAG derived from TA simulates the eAG representation of X ¹⁶.

The above provides a sketch of proof for the following connexion between interpre-

¹⁵The following sketch leaves recursive element definition out.

¹⁶We obliterate the question of node types here. We only compare the bare simulation and interpretation relations.

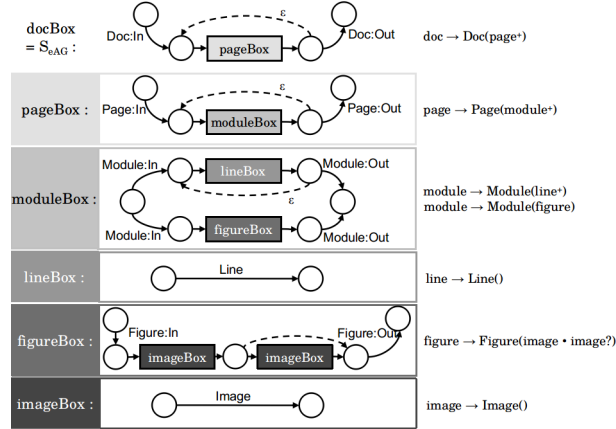


Figure 5.7: For all $n \in N$ as defined in Figure 5.6, each $n\text{Box}$ representation (middle) and $R_n \subset R$ (right).

tation and simulation for validation:

Property 5.2.1 Be a tree X and a tree automaton TA . Be G_X the eAG representation of X ; be S_{TA} the SeAG derived from TA . Then, if there is an interpretation of X by TA , then S_{TA} simulates G_X .

Nota. A complete proof can be consulted in the Appendix 12 (in French).

5.2.3 Precisions on SeAG

At this point, we have defined eAG, an annotation model compatible with the expression of multilayer, cyclic annotation, together with SeAG, an appropriate schema mechanism. The relation between an eAG and a SeAG is a certain kind of simulation, in which the roots and the leaves of the schema and the instance are involved. We want to investigate some properties of SeAG further here.

First, we observe that, for a given annotation language, several schemas could be defined. We will try to show that, if from a static point of view, those different schemas are hard to discriminate, some present better properties, qualitatively speaking, in the perspective of being amended.

Second, we will consider the notion of redundancy, characterizing the fact some nodes or edges of a schema may be deleted or merged without this altering the language of the schema [128].

5.2.3.1 Multiple forms for the same schema?

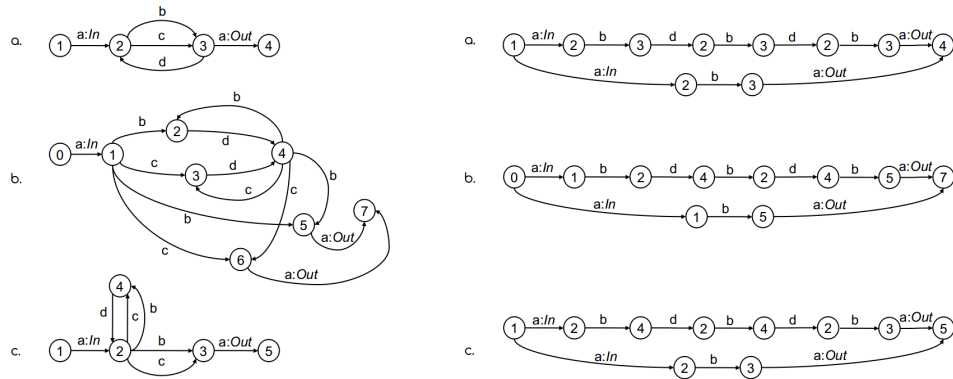
Let us suppose, for didactic purpose, that an editor possesses an initial vision of the annotation to be done in terms of language of annotation, and thus, we consider here that the editor abstracts that language of annotation by means of a set of regular

expressions of the kind that were illustrated in Paragraph 5.2.1.3, that she then wants to translate into a schema¹⁷.

It is a well known result that one regular expression may translate into a whole diversity of automaton. A whole field of the literature precisely investigates how to construct an automaton from a regular expression (e.g. [128], [84], [25], [30], [99], [5]). It appears that all those constructions are compatible with the eAG/SeAG model, and that, from a static point of view, none in particular would have to be favoured.

Let us take an example. Be the alphabet $\Sigma = \{a, b, c, d\}$. Be a regular expression over that alphabet $R = a : In \cdot [(b|c)d]^*(b|c) \cdot a : Out$.

This regular expression can be translated into several automata, according to the transformation method operated, as indicated below (left column). The first automaton representing R (a.) is a deterministic automaton with a minimal number of edges. The second is the Glushkov [84, 30] automaton associated to R . The last is the des-epsilononised version of the Ott automaton for R . On the right column, we represent three eAG instances, one for each schema, that all express exactly the same annotation.



From this point of view, the three schemas seem to be equally good. Yet, one may notice that, in the perspective of being modified, not all three are equal, in the sense that, for a given amendment to the regular expression, not all schemas will have to be modified by the same sequence of operations.

For example, let us consider that the editor first modelled her corpus by means of the regular expression $R = a : In \cdot [(b|c)d]^*(b|c) \cdot a : Out$, that she then translated into one of the above schemas (a), (b) or (c). Let us now imagine that she wants to change the annotation language of the schema, so that it represents the new regular expression $R' = a : In \cdot [(b|c)d]^*(X|c) \cdot a : Out$. This means that, when adding indexes to the symbols of the original and the updated regular expressions, $R = a : In_1 \cdot [(b_2|c_3)d_4]^*(b_5|c_6) \cdot a : Out_7$ shall become $R^b = a : In_1 \cdot [(b_2|c_3)d_4]^*(X_5|c_6) \cdot a : Out_7$. In other words, naïvely speaking, in terms of schema transformations, it means that the label of all the edges of the SeAG that represent the symbol indexed 5 in R shall

¹⁷We consider this hypothesis editor modelling a literary corpus by means of regular expressions for didactic purpose only. In particular, the aim of the following discussion is not to establish how a schema could automatically be derived from a set of regular expressions.

be replaced by X . Yet this strategy cannot be undergone indifferently on the three schemas:

- *Schema (c.)* : The easiest case is this one, since there is a one-to-one correspondence between the indexed symbols of R^b and the edges of the schema. The transition from $S_c(R)$ to $S_c(R')$ can be done by relabelling the edge e of label b between the nodes typed 2 and 3. Propagating this amendment to the instances of S_c may then be done by relabelling each e_{2b3} in an eAG defined by $\delta_E(e_{2b3}) = e$.
- *Schéma (b.)* : Schema (b.) is different, in that there is no such correspondence between the indexed symbols in R^b and the edges of the schema: in particular, the symbol with 5 as an indec is represented by two edges in the schema, namely $4[b]5$ et $1[b]5$. Operating the change on the schema and propagating it to the instances shall thus mean to relabel those two edges and the edges of the instance that instantiate them..
- *Schéma (a.)* : Schema (a.) is the most interesting. In this case, there is no edge representing the symbol indexed 5 from the regular expression specifically. Indeed, the edge $2[b]3$ represents the indexed symbols b_2 and b_5 from R^b . The strategy to go from $S_a(R)$ en $S_a(R')$, is thus far more complicated¹⁸.

Actually, in the above example, what the amendment to the annotation language semantically means is that “a label b ending a sequence $(b|c)^+$ sets apart from the other b symbols”, and thus shall, it and not the other bs , be turned into an X : it is the existence of this distinction among the otherwise identical bs , that only exists *in the editor’s mind*, that drive her to want to replace that special b and not the others. One could say that there are *semantically dependent* indexed symbols in a regular expression, in the sense that they are distinct symbols that represent non-distinguishable entities, and *semantically independent* labels, whose letter of the alphabet is the same but that, in the mind of the editor, are somehow different (and thus might be amended independently). Yet the status of each symbol is not initially given: as illustrated above, it may get revealed... when attempting at modifying the regular expression.

In that perspective, even if the schema (c.), because it turns here to associates an edge to each semantically independent symbol of the regular expression, seems to be the most promising. yet since the existence of semantical dependencies among the symbols seems not to be given a priori, it can hardly be exploited to choose among several schemas in order to optimize schema transformability. It should also be noted that a transformation may precisely consist in making two labels independent (or dependent), which is what happens in this example when considering schema (a.).

As a consequence, we will have to do with the fact several schemas may equally be chosen for modelling the same corpus. Choosing among those schemas will then be a question of taste or luck for the editor – until further research on this intriguing point.

¹⁸It may consist in creating an additional node typed 5 and to substitute, in the schema and in the instances of the schema, the paths (instantiating) $2[b]3[a : Out]4$ by paths $2[X]5[a : Out]4$, without impacting the other paths: then, the new nodes typed 5 shall be given the reference value of the nodes typed 4 they replace...

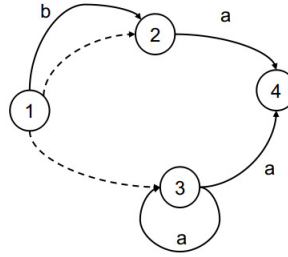


Figure 5.8: A non-redundant but ambiguous graph.

Noteworthy still, we will see in the Part IV of this dissertation that, when considering certain tasks (the schema-aware parsing of an inline markup syntax for eAG), there is a certain schema construction that has better algorithmic properties than the others.

5.2.3.2 Redundancy and ambiguity

As a conclusion, it appears that the shape of schemas is quite open. Yet another aspect in which two schemas that express the same annotation language may not be equal is how redundant and how ambiguous they are.

Intuitively speaking, redundancy qualifies the fact several nodes and edges of a schema may be deleted or merged without this affecting the language of the schema, and a schema is unambiguous “if for each word w , there is at most one path through the state diagram of M that spells out w ” [30]. Those are two different characteristics, in the sense a schema may be non redundant while presenting some ambiguity. As an example, consider the schema from Figure 5.8. Deleting any of the edges would impact the language of the graph, and merging the nodes 2 and 3 also (then the word *baa* would be part of the language, which it is not initially), so the graph is not redundant; on the contrary, the word *a* may result from the execution of two different paths.

Redundancy is not a desirable quality for a schema, particularly in the perspective of it being amendable: if two paths from the schema define exactly the same annotation language, which is not useful in general, then if the amendment aims at modifying the part of the language of the schema that corresponds to those paths – that is, at preventing from the expression of words from the language of those paths and replacing those by other words –, then the two paths shall be modified synchronously.

Non ambiguity, for its part, is a required characteristics for some other schema languages like DTD. Additionally, based on the traditional definitions of redundancy and ambiguity (to be given below), non-ambiguity implies non-redundancy. Still, imposing SeAG to be non ambiguous in the traditional sense is not a reasonable restriction. In particular, the annotation language of the graph on Figure 5.8 cannot be expressed but by ambiguous graphs, as in fact any language in the form $R = R' \cdot A | A' \cdot R''$ where $lang(A) \cap lang(A') \neq \emptyset$.

We thus propose an alternative definition of ambiguity, called *typed ambiguity*, consistent with our graph model, resulting in a preserved expressivity of SeAG. A SeAG

schema will then ideally have to be non redundant and non-ambiguous in the sense of this new definition.

Ambiguity can be defined as follows.

Definition 5.2.3.2.1 : FSM ambiguity. An FSM is unambiguous iff for all word m in its language, there is at most one way to browse through the FSM along a path whose label sequence spells m .

Definition 5.2.3.2.2 : Regular expression ambiguity. An FSM is unambiguous iff for all word m in its language, there is at most one execution of the word m .

As indicated above, this notion is very restrictive. The notion originates from the theoretical work of Book on regular expressions [25]: from the consideration that some words from the language of some regular expressions may result from different executions of the expression, the authors propose a method to test whether a regular expression is ambiguous or not. Later, ambiguity has been adapted and discussed widely in the literature on SGML (e.g. [30], [32], [185]) and XML ([89], [80]). In the context of SGML, non-ambiguity is required for DTD grammars for several reasons that, in the end, one may hardly find consistent. Hence the question whether this notion is relevant or not for SeAG.

Among the justifications for imposing unambiguity for DTD, it has widely been suggested that, in the context of SGML, “[u]nambiguity [was] intended to make SGML document grammars easier to read by humans. It is questionable, though, whether this goal is really achieved.” [31]. Another justification for the recourse to unambiguous grammars is algorithmic: “The intent of the authors of the [DTD] Standard is twofold: They wish to make it easier for humans to write and understand expressions that can be interpreted unambiguously and, at the same time, to provide a means of generating parsers that do not have exponential blow up in size.” [185]. Those reflections, that date back to the origins of SGML validation by schemas, rest upon the following pragmatic considerations: [88] had defined a parser generator for which non ambiguous grammars were well-adapted – and that generated parsers that, in turn, could be used to validate SGML data, that is, to assess whether a document was conform to a given grammar or not. Still, it seems that the limitation to non-ambiguous grammars can be bypassed to that purpose [115] – and actually, RELAX NG does not restrain the ‘content models’ that define the XML grammar to be unambiguous regular expressions [48].

In the end, unambiguity for SGML/XML grammars seems have mostly historical and pragmatic grounds. None of the above justifications applies to SeAG, for which we will thus not impose any restriction regarding ambiguity.

Still, it seems interesting enough to mention that the eAG/SeAG model, without any additional restriction, is naturally non-ambiguous in some sense, in that given an eAG validated by a SeAG, it is always possible to know without any... ambiguity which of the subgraph of the SeAG is instantiated by any path from the eAG. This characteristic is, as we will see, of crucial importance for the derivation of a transformation on eAG from an amendment operated on a SeAG, as one may imagine intuitively. We propose to formalize this characteristics of the eAG/SeAG model by means of the notion of typed ambiguity.

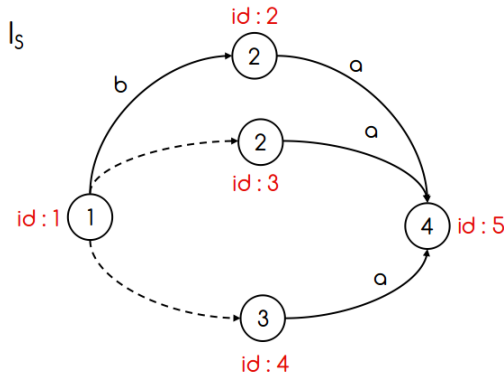
Definition 5.2.3.2.3 : typed label. Be $S = ((V, E), label, ref, type, id)$ a SeAG. Be $label_{\mathcal{T}} : E \rightarrow \mathcal{T} \times \mathcal{L} \times \mathcal{T}$ the function that associates to each edge of X the triple constituted of the type of its summit, its label and the type of its end.

Property 5.2.3.2.1 For any $S = ((V, E), label, ref, type, id)$, $label_{\mathcal{T}}$ is injective. Thus the automaton defined by $X = (V, E, type, label_{\mathcal{T}}, ref, ind)$ is unambiguous.

Preuve. The function is indeed injective since no two nodes can have the same type in a SeAG, and no two edges between the same pair of nodes can have the same label. Thus, in the FSM obtained by replacing the labels, in S , by the $label_{\mathcal{T}}$, no two transitions are identical, which is an obvious case of unambiguous FSM.

Definition 5.2.3.2.4: typed ambiguity. $S = ((V, E), label, ref, type, id)$ is typed-unambiguous iff the FSM defined by $G' = (V, E, label_{\mathcal{T}}, type)$ is non ambiguous.

Let us now take an example to illustrate that notion. Consider the SeAG given in Figure 5.8 page 113, and the following corresponding eAG I_S :



We have already indicated that the schema is ambiguous in the traditional sense; still, as any SeAG, it is typed-unambiguous. And indeed, if one considers the instance I_S , even if two paths from that instance spell the same, it is very clear that the path spelling **a** to the node whose $id = 3$ instantiates the path $ch_1 = 1[\epsilon]2[a]4$, from the schema, while the second path of I_S spelling **a** instantiates $ch_2 = 1[\epsilon]3[a]4$. This information is interesting in the sense that it permits to distinguish ch_1 and ch_2 , based on a finer criterion than the sequence of labels.

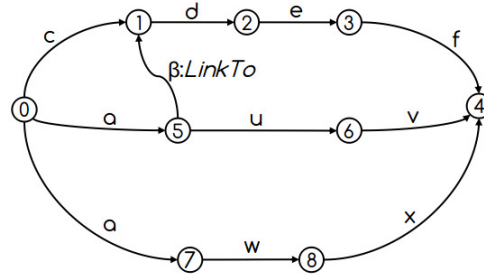
Let us now come to the notion of redundancy, that qualifies the fact that certain parts of the language of the annotation (words, suffixes and prefixes of words) are defined several times identically in a schema. This notion has been treated by Ott for FSM [128] in a very interesting way. Ott breaks the general notion of redundancy into three notions: equivalent nodes; nodes that can be merged and superseding nodes.

Definition 5.2.3.2.4 : Equivalent nodes. Two nodes of a FSM are equivalent if and only if, for each input symbol, both lead to the same or equivalent nodes.

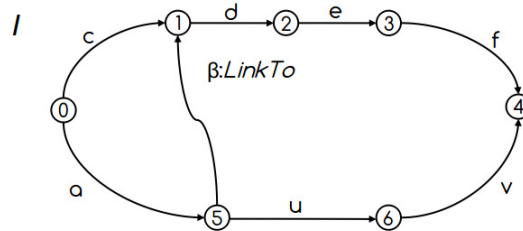
Commentaire. The equivalence relation above is actually identical to the maximal self bisimulation of a graph. In Ott's view, two equivalent nodes can be merged without this altering the language of the FSM. In SeAG, we will not use this notion for that purpose.

Definition 5.2.3.2.5 : Etats fusionnables. Two nodes v_1 et v_2 of a FSM can be merged iff for any configuration, (S, u) , being in one implies being in the other: $\langle v_1 \rangle \Leftrightarrow \langle v_2 \rangle$.

Commentaire. Identically, for Ott, who deals with FSM properly speaking, for which the language is the only aspect that counts, the FSM can be transformed by merging those nodes. In our case, this does not apply, because of the particular edge semantics we have defined for SeAG. Consider the following schema below:



For instance, if we do merge the nodes 5 et 7, the schema that we get will validate the eAG I below, which the original schema did not:



Definition 5.2.3.2.6 : Superseding nodes. Be a FSM $G = (V, E)$. A node $v_1 \in V$ supersedes $v_2 \in V$ iff the following conditions hold:

1. For any configuration, (S, u) , $\langle v_1 \rangle \Leftarrow \langle v_2 \rangle$;
2. For any input symbol, v_1 leads either to the nodes v_2 leads to, or to nodes that supersede them.

We will denote the fact $v_1 \succ v_2$ the fact v_1 supersedes v_2 .

By extension, we define the operator \succeq , for the "equals or supersedes" relation, given any node supersedes itself.

Interpretation of the \succeq relation. Be $G = (V, E)$ a Schema. Formally, the above relation can be rephrased as follows.

$$[1] v_1 \succeq v_2 \Leftrightarrow \begin{cases} \text{a. } \langle v_2 \rangle \Rightarrow \langle v_1 \rangle \\ \text{b. } \forall e_2 \in E, v'_2 \in V \mid v_2[e_2]v'_2 \subseteq G, \\ \quad \exists e_1 \in E, v'_1 \in V \mid \text{label}(e_1) = \text{label}(e_2) \wedge v_1[e_1]v'_1 \subseteq G \wedge v'_1 \succeq v'_2 \end{cases}$$

Be $follow(v)$ the function that to v associates the set of nodes that can be attained from v by following the paths included in G (v included). Be $\iota : v_i \mapsto \bigcup_{v \in follow(v_i)} \left(\bigcup_{X \subseteq_v^{root(G)}} \mathcal{L}(X) \right)$.

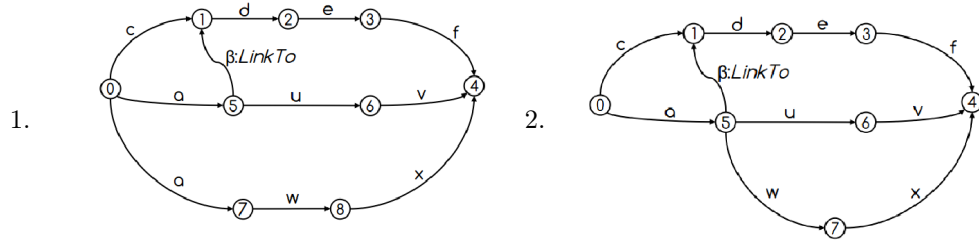
One may check that $v_1 \succeq v_2 \Rightarrow \iota(v_2) \subseteq \iota(v_1)$. This is due to the fact that :

- $(\langle v_2 \rangle \Rightarrow \langle v_1 \rangle) \Rightarrow \forall G_2 \subseteq_{v_2}^{root(G)} G, \exists G_1 \subseteq_{v_1}^{root(G)} G$ tel que $\mathcal{L}(G_2) \subseteq \mathcal{L}(G_1)$
- $v_1 \succeq v_2 \Rightarrow \forall v'_2, e_2 \mid v_2[e_2]v'_2 \subseteq G, \exists v'_1 \mid (\langle v'_2 \rangle \Rightarrow \langle v'_1 \rangle) \wedge v'_1 \succeq v'_2$.

Recursively, we can establish the above result.

The above means that the \succeq relation may be used to identify the connected subgraphs of a schema, starting at the root of the schema and ending at its leaf, whose languages are included one into the other, and that simulate one another.

Indeed, the fact two subgraphs share the same language is not enough for there being a \succeq relation. Consider the fact that among the two graphs, that have exactly the same language, but if the root of the first one supersedes the root of the second, the converse is not true.



Actually, the nature of the structural similarities required between two graphs for the root of one to supersede the root of the other can be qualified easily: retrieving the condition a. from the above reformulation [1] of the \succeq relation yields the definition of a... simulation.

We thus propose to use redundancy as an equality relation between schemas.

Definition 5.2.3.2.7 : SeAG equality. Two schemas will be considered equal iff their roots mutually supersede one another.

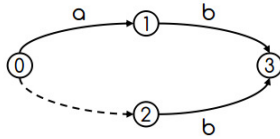
Comment. This equality relation is far more satisfying than what could have been expected to be a natural equality relation, in a context where simulation is used to

compare graphs, namely, bisimulation. Given D a simulation of A by B , there is a bisimulation between A and B iff D^{-1} defines a simulation. Actually, an equivalence relation can be built onto bisimulation and used as an equality relation that is far easier to compute than, for instance, isomorphism [93]: two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ shall be considered “equal” “égaux” iff all the nodes of each graphs are involved in the maximal bisimulation between the two graphs, and so that the pair of their roots belong to the bisimulation. The non-redundancy of a schema shall then be defined as the fact that the maximal self bisimulation of that schema shall be the identity relation.

Yet, the notion of (bi)simulation and supersedure differ on an essential point, that matters for the identification of redundancy. While (bi) simulation investigates each pair of nodes in the graphs, supersedure only considers pairs of nodes (v_1, v_2) so that $\langle v_2 \rangle \Rightarrow \langle v_1 \rangle$, that means, that are preceded, in the schema, by some linear paths that spell the same. This is severe pruning. In other words, when calculating the self bisimulation of a graph, any node of the graph is considered an input node. This prevents from comparing the upwards context of those nodes, i.e. from comparing the sequence of the labels that lead to those nodes: comparison is done downwards only, in the direction of the leaf of the graph. Supersedure considers that the input nodes of a FSM (or here, of a schema) are well-defined: they are nodes explicitly defined as the input nodes (or the root of the SeAG here).

To illustrate the difference, and why supersedure is better for the definition of equality, consider the following example.

Be S the following schema.



The maximal simulation relation D and supersedure relation R are the following:

- $D = \{(0; 0), (1; 1), (2; 2), (3; 3), (0; 1), (1; 2), (2; 2)\}$;
- $R = \{(0; 0), (1; 1), (2; 2), (3; 3), (0; 1)\}$.

hence the maximal bisimulation relation B_D and bi-supersedure relation B_S :

- $D = \{(0; 0), (1; 1), (2; 2), (3; 3), (1; 2)\} \neq Id$;
- $R = \{(0; 0), (1; 1), (2; 2), (3; 3)\} = Id$.

Thus, if equality is defined in terms of bisimulation, the schema is considered redundant, while it is not if one considers bi-supersedure as an equality relation.

5.3 SeAG Validation: A *Posteriori* and On-the-fly Validations

Validation is the process by which it is assessed whether an annotation is conform to a certain schema. This assessment can be operated in two different contexts: first, the annotation is done regardless of a schema, and is then validated against a schema that has been independently defined; second, a schema is provided to the editor when she starts annotating. In this case, validation may not be done a posteriori. On the contrary, the schema may be used, in the course of the eAG manufacturing, as a guide for annotating the data, just the way XML schemas can be used in an XML editor featuring content assist.

The first case will be referred to as a posteriori validation. The second, as on-the-fly validation. We show how both can be done with SeAG, and analyze the properties of both in the following.

5.3.0.3 A Posteriori Validation

Validating is defined as follows :

Definition 5.2.16: validation. Be S a SeAG and G an eAG. The S validates G iff there is a retyping of G so that $S \xrightarrow{typ.} G$, so that the pair of the roots of S and G and the pair of their leaves belong to the maximal node-typed simulation.

It is quite clear that given a schema S and an eAG that S validates, changing the type of one node of G may prevent G from being valid while the structure and the semantics of G have not changed. This means, conversely, that an eAG whose types do not match the ones of a schema is not necessarily ‘wrong’: it may just need retyping for being valid¹⁹.

In the context of a posteriori validation, there is little chance the types of the instance and of the schema shall match, since both were built independently. Retyping is thus needed. Yet, there is no need for calculating this retyping, given the following result:

Property 5.2.6 Be two directed, labelled graphs $A = ((V_A, E_A), type)$ and $B = ((V_B, E_B), type)$ whose nodes are typed over two type set \mathcal{T}_A and \mathcal{T}_B , so that $B \xrightarrow{strong} A$. Then there is a retyping of the nodes of A so that $B \xrightarrow{typ.} A$.

Proof. Let us denote D the strong simulation of A by B . It suffices to define the retyping function $retype : \mathcal{T}_A \rightarrow \mathcal{T}_B$ so that $\forall (v_A, v_B) \in D, retype(type(v_A)) = type(v_B)$.

Hence a posteriori validation can be performed as follows:

¹⁹The reason why validation is defined in terms of the more exclusive notion of node-typed simulation is that it permits on-the-fly validation, as will be illustrated in the next paragraph.

Definition 5.2.17: a posteriori validation. Be S a SeAG and G an eAG. Then S validates G iff $S \xrightarrow{\text{strong}} G$, so that the pair of the roots of S and G and the pair of their leaves belong to the maximal strong simulation.

Calculating the maximal (strong) simulation of one graph by another one has been investigated already [129, 4, 81, 144]. The main result is the following:

Property 5.2.3 Be a SeAG S and an eAG G . Calculating the maximal simulation of G by S performs in $O(|V \cup V_S| \cdot |E \cup E_S|)$ [144].

Comment. This is a reasonable cost for a cyclic graph-based data model.

5.3.0.4 On-the-fly Validation

Now, let us consider the case where a schema is provided before starting annotating, and where the produced data has to be valid. Checking algorithmically whether a graph is valid or not could be tricky. Instead, we propose a special case of on-the-fly validation, that is, validation by construction. We show here that there is a matrix representation for SeAG and eAG so that, given the representation of a schema, only valid instances can be represented. hence the qualifier “by construction”.

Definition 5.2.18 (Identifier sets) Be a graph $G = (V, E)$. There are two countable ordered sets \mathcal{I}_G and \mathcal{J}_G and two bijective functions $id : V \rightarrow \mathcal{I}_G$ and $id : E \rightarrow \mathcal{J}_G$ identifying the nodes and edges of G . The i^{th} element of the set \mathcal{I}_G , for instance, is denoted $[\mathcal{I}_G]_i$.

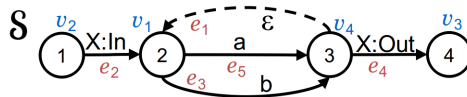
Definition 5.2.19 Be a graph $G = (V, E)$, \mathcal{I}_G and \mathcal{J}_G two sets of identifiers. Provided G contains no connected subgraph limited to a node and no loop, G can be represented by its incidence matrix $[G]^{\mathcal{I}_G, \mathcal{J}_G}$ so that, $\forall (i, j) \in \mathcal{I}_G \times \mathcal{J}_G$:

$$\begin{aligned} [G]_{i,j}^{\mathcal{I}_G, \mathcal{J}_G} &= 1 \text{ iff } \exists v[e]v' \subseteq G; id(v) = i \wedge id(e) = j \\ &= -1 \text{ iff } \exists v[e]v' \subseteq G; id(v') = i \wedge id(e) = j \\ &= 0 \text{ else.} \end{aligned}$$

Property 5.2.4 Be a SeAG $S = (V_S, E_S)$. Ordering the sets $\{type(v); v \in V_S\}$, $\{(type(v), label(e), type(v')); v[e]v' \subseteq S\}$, provides two special node and edge identifier sets \mathcal{T}, \mathcal{X} .

Proof. In a SeAG, no two nodes have the same type or are connected by two edges with the same label (Def. 5.2.9). Any ordering of the sets is fine.

Discussion (1). Consider the following SeAG :



The values v_i are possible identifiers for the nearby nodes, and e_j for edges, so that $\mathcal{I}_S = [v_1, v_2, v_3, v_4]$, for instance. Then, Property 5.2.4 means that it is possible to represent the incidence matrix of an SeAG S by indexing lines and columns either on any $\mathcal{I}_S \times \mathcal{J}_S$ or on $\mathcal{T} \times \mathcal{X}$ in particular. For instance, when indexed over $\mathcal{T} \times \mathcal{X}$:

$$[S]^{\mathcal{T}, \mathcal{X}} = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \text{1X:In}_2 \\ \text{2a}_3 \\ \text{2b}_3 \\ \text{3e}_2 \\ \text{3X:Out}_4 \end{array} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 1 & -1 & 0 \\ 0 & -1 & -1 & 1 & 1 \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

It is also possible to express a *subgraph* of S in an incidence matrix indexed over the full identifier sets. For instance, below is the incidence matrix over \mathcal{I}_S and \mathcal{J}_S of $H = \{v[e]v' \subseteq S; (\text{type}(v), \text{label}(e), \text{type}(v')) = (2, b, 3)\}$, subgraph of S :

$$[H]^{\mathcal{I}_S, \mathcal{J}_S} = \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \end{array} \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

Definition 5.2.20 Given a $n \times m$ matrix $[M]$ of integers, the positive restriction of $[M]$ is the $n \times m$ matrix $[M^+]$ so that $\forall i, j$, $[M^+]_{i,j} = [M]_{i,j}$ iff $[M]_{i,j} > 0$, else $[M^+]_{i,j} = 0$. The definition of $[M^-]$ the negative restriction of $[M]$ is natural.

Discussion (2). Consider two graphs G and H , $H \subseteq G$ and the incidence matrix $[H]^{\mathcal{I}_G, \mathcal{J}_G}$. Then the positive restriction of $[H]^{\mathcal{I}_G, \mathcal{J}_G}$, read column by column, lists the identifiers of the nodes that are the summits of the edges of H whose identifier matches the one of the column. Conversely, the negative restriction of $[H]^{\mathcal{I}_S, \mathcal{J}_S}$ defined in the Discussion (1) page 120 is:

$$[H^-]^{\mathcal{I}_S, \mathcal{J}_S} = \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \end{array} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

Note that the sum of the positive and negative restriction of any incidence matrix gives the incidence matrix.

Definition 5.2.21 (Template) Be S a SeAG, G a graph that can be represented by its incidence matrix, and $\mathcal{I}_G, \mathcal{J}_G$ identifier sets for G . Consider the block-matrix obtained by replacing each value $s_{i,j}$ of $[S]^{\mathcal{T}, \mathcal{X}}$ by a matrix $[M_{i,j}]$, so that:

- $s_{i,j} = 0 \Rightarrow [M_{i,j}] = [\emptyset]^{\mathcal{I}_G, \mathcal{J}_G}$, where \emptyset is the empty graph, whose incidence matrix is always zero ;
- $s_{i,j} = 1 \Rightarrow [M_{i,j}] = [A]$, where $[A]$ is the positive restriction of the incidence matrix over $\mathcal{I}_G, \mathcal{J}_G$ of $H_j \subseteq G$, with $H_j = \{v[e]v' \subseteq G; (\text{type}(v), \text{label}(e), \text{type}(v')) = [\mathcal{X}]_j\}$;
- $s_{i,j} = -1 \Rightarrow [M_{i,j}] = [B]$, where $[B]$ is the negative restriction of the incidence

matrix over $\mathcal{I}_G, \mathcal{J}_G$ of H_j .

This block-matrix is called the expression of G on the template of S , denoted $[G/Temp.S]$.

Example. Consider the SeAG S defined in the Discussion (1) page 120. The expression of S on its own template is:

$$[S/Temp.S] = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{c} \mathbf{1X:In}_2 \\ \mathbf{2a}_3 \end{array} \\ \begin{array}{c} \mathbf{2b}_3 \\ \mathbf{3\epsilon}_2 \end{array} \\ \begin{array}{c} \mathbf{3X:Out}_4 \\ \mathbf{4} \end{array} \end{array} \begin{bmatrix} [A_1] & 0 & 0 & 0 & 0 \\ [B_1] & [A_2] & [A_3] & [B_4] & 0 \\ 0 & [B_2] & [B_3] & [A_4] & [A_5] \\ 0 & 0 & 0 & 0 & [B_5] \end{bmatrix}$$

with, for instance, $[A_3] = [H^+]^{\mathcal{I}_S, \mathcal{J}_S}$ and $[B_3] = [H^-]^{\mathcal{I}_S, \mathcal{J}_S}$ as defined in Discussion (2) page 121.

Definition 5.2.22 Be S an SeAG and $G, \mathcal{I}_G, \mathcal{J}_G$ a graph containing no subgraph limited to a node and no loop, along with two sets of identifiers. G is said to be fully expressible on the template of S , denoted $G \triangleleft [Temp.S]$, iff the sum of the inner matrices of $[G/Temp.S]$ is equal to $[G]^{\mathcal{I}_G, \mathcal{J}_G}$ the incidence matrix of G , indexed over the same sets as the inner matrices of $[G/Temp.S]$.

Property 5.2.5 Be S an SeAG. Then S is fully expressible on its own template.

Proof. $\forall l \in [0; |\mathcal{X}|[$, the l^{th} column of $[S/Temp.S]$ contains two matrices $[A_l]$ and $[B_l]$. Since they are respectively the positive and negative restrictions of the incidence matrix of $H_l \subseteq S$, which is the union of all the subgraphs $v[e]v'$ characterized by the same triple $[\mathcal{X}]_l$ of types and label, $[A_l] + [B_l] = [H_l]^{\mathcal{I}_S, \mathcal{J}_S}$. Since \mathcal{X} is the set of possible triples for S , $\sum_{0 \leq l < |\mathcal{X}|} [H_l]^{\mathcal{I}_S, \mathcal{J}_S} = [S]^{\mathcal{I}_S, \mathcal{J}_S}$.

Importantly, only schemas define a template. In particular, given an instance G and any identifier sets \mathcal{I}, \mathcal{J} , since there may not be bijections between those sets and \mathcal{T}, \mathcal{X} , the notion of template of G is undefined. Still, it is possible to try to express G , not over its own template, *but over the template of a given schema S* .

Let us denote this representation $[G/Temp.S]$. The schema defines the template, that is, the outer matrix of $[G/Temp.S]$, indexed over $\mathcal{T} \times \mathcal{X}$: it restricts the types, the labels between two given types and the paths along which those labels may occur. Be then $[\mathcal{X}]_l \in \mathcal{X}$. Just like above, we can define $H_l = \{v[e]v' \subseteq G; (type(v), label(e), type(v')) = [\mathcal{X}]_l\}$, so that $H_l \subseteq G$. Then the inner matrices of $[G/Temp.S]$ are defined just the same way as those in $[S/Temp.S]$, that is: in the l^{th} outer column, on the right outer lines, as the positive and negative restrictions of $[H_l]^{\mathcal{I}, \mathcal{J}}$ (see Definition 5.2.21).

Interestingly, this approach can be taken for any graph G and any schema S . If the graph contains no edge conforming the schema, then $[G/Temp.S]$ is null. On the contrary, an important result is that provided G is an instance of S , then G is *fully expressible* on $[Temp.S]$. We can even go further:

Property 5.2.6 Be a SeAG S and an eAG G . Then S validates G iff $G \triangleleft [Temp.S]$ and $type(leaf(G)) = type(leaf(S))$ and $type(root(G)) = type(root(S))$.

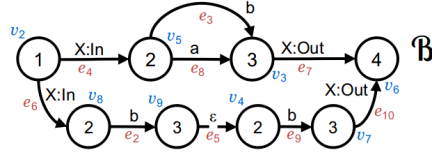
Proof. \Rightarrow : S validates G , then the types of the two graphs' leaves are equal, by definition of validation. Idem for the roots. The fact that validation implies $G \triangleleft [Temp.S]$ can be proven just like Property 5.2.5, with one more argument. The fact that the sum of the two inner matrices characterised by the same $L \in \mathcal{X}$ yields the incidence matrix of the union of all the subgraphs $v[e]v' \subseteq G$ characterised by L holds. Yet, it has to be proven that there is no subgraph $v[e]v' \subseteq G$ so that $(type(v), label(e), type(v')) \notin \mathcal{X}$. Since G is rooted and connected, one can check that the presence of such a subgraph shall contradict the existence of a rooted simulation.

\Leftarrow : Be $G = (V, E)$, $S = (V_S, E_S)$. $G \triangleleft [Temp.S]$ implies that $\forall v[e]v' \subseteq G$, $\exists L \in \mathcal{X}$ so that $(type(v), label(e), type(v')) = L$, which means $\exists! v_S[e_S]v'_S \subseteq S$ so that $type(v_S) = type(v)$, $type(v'_S) = type(v')$ and $label(e_S) = label(e)$.

This defines two functions $\delta : V \rightarrow V_S$ and $\delta_E : E \rightarrow E_S$ so that $\forall v[e]v' \subseteq G$, $\exists! (\delta(v), \delta_E(e), \delta(v')) \in V_S \times E_S \times V_S$ so that $\forall x$, $type(\delta(x)) = type(x)$, $\forall y$, $label(\delta_E(y)) = label(y)$ and $\delta(v)[\delta_E(e)]\delta(v') \subseteq S$.

Additionally, the fact that $type(root(G)) = type(root(S))$ implies $\delta(root(G)) = root(S)$. Then $D = \{(v, \delta(v)); v \in V\}$ is a rooted, node-typed simulation of G by S .

Illustration part 2. Consider the eAG B from Example 5.2.2.3. Let us equip its nodes and edges with identifiers, as shown below.



The representation of B in $[Temp.S]$ is:

$$[B/Temp.S] = \begin{array}{c} 1 \quad \begin{array}{c} \text{X:In}_2 \\ [A_1] \end{array} \quad 2a_3 \quad 2b_3 \quad 3e_2 \quad 3\text{X:Out}_4 \\ 2 \quad \begin{array}{c} [B_1] \\ [A_2] \end{array} \quad [A_3] \quad [B_4] \quad 0 \\ 3 \quad 0 \quad [B_2] \quad [B_3] \quad [A_4] \quad [A_5] \\ 4 \quad 0 \quad 0 \quad 0 \quad 0 \quad [B_5] \end{array}$$

with $[B_3]$ the negative restriction of $[H]^{\mathcal{I}_B, \mathcal{J}_B}$, for instance, for $H = \{v[e]v' \subseteq B; (type(v), label(e), type(v')) = (2, b, 3)\}$:

$$[B_3] = \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \end{array} \begin{array}{c} e_1 \quad e_2 \quad e_3 \quad e_4 \quad e_5 \quad e_6 \quad e_7 \quad e_8 \quad e_9 \quad e_{10} \\ \left[\begin{array}{cccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right] \end{array}$$

Now compare $[B/Temp.S]$ with $[S/Temp.S]$ as detailed in the Example page 122. The two matrices share the same outer matrix, which is descriptive of S , they only differ by the values of the inner matrices (e.g. see the value of $[B_3]$ for S in Example 5.3.0.4). Based on Property 5.2.6, we can finally conclude:

Given a schema S , the eAGs it validates are the well-formed eAGs model that can be fully expressed in $[Temp.S]$, and whose root and leaf types respect those of S . This means that an instance of S is an eAG *that can be described by* the set of matrix values that fill $[Temp.S]$. From a manufacturing point of view, if the annotator of a resource is given means (through an ergonomic HCI) to define the matrix values corresponding to $[Temp.S]$, in a way that ensures well-formedness, then, by construction, the resulting graph will be valid against the schema. This meets the goal of providing on-the-fly validation for M-S data.

5.4 Conclusion

In this chapter, we introduced eAG, an extension of Annotation graphs, along with a novel schema model based upon the notion of simulation. A dedicated representation for eAGs and schemas enables to proceed to validation “by construction”: provided a schema, only valid eAGs can be expressed, which bypasses the algorithmic cost of traditional approaches for validation of graph-structured data.

Still, the eAG data model is not restricted to this use case, and simulation-based validation can be adapted to the situations where any eAG $G = (V, E)$ is confronted to any SeAG $S = (V_S, E_S)$. First case, G was made according to a schema $S' = (V_{S'}, E_{S'})$, and the question is whether it conforms to S or not. By transitivity of simulation, S validates G iff S simulates S' so that $(leaf(S'), leaf(S))$ are in the simulation (indicating, *modulo* retying the nodes of S , a node-typed simulation of S' by S). This checks in $O(|V_{S'} \cup V_S| \cdot |E_{S'} \cup E_S|)$ [144]. Second case, G was not made according to any schema. In this case, node types are irrelevant. An adaptation of SeAG validation is: S validates G iff there is a (general) simulation $D \subseteq V \times V_S$ so that $\forall v \in V, \exists! v_S \in V_S$ so that $(v, v_S) \in D$ (the uniqueness of v_S for each v defines a typing of the nodes of G according to S). This checks in $O(|V \cup V_S| \cdot |E \cup E_S|)$. In both cases, this is a reasonable cost for a cyclic graph-based data model.

Part III

Linear Extended Annotation Graphs: an Inline Markup Syntax for eAG

Chapter 6

Inline Multilayer Annotation

Linear extended Annotation Graphs (LeAG) is an inline markup syntax for eAG. The purpose of LeAG is to enable the expression of eAG annotations by means of any notepad application, in a human-readable form. LeAG must therefore: 1) support unambiguous translation into the eAG syntax, and 2) enable to represent, by means of tags, multilayer, cyclic annotation.

The first section of this chapter is a theoretical discussion about the hybrid nature of the LeAG markup, between the inline and stand-off paradigms, which will lead to the formulation of an equivalence relation for LeAG documents.

We then introduce, step by step, the LeAG syntax: we gradually show how to represent the different bricks eAGs are made of in a markup manner: hierarchies, multitrees (and goddags), attributes, links and quotes. We then interrogate the correspondence between eAG and LeAG, in order to establish the parsability of LeAG into eAG.

6.1 Introduction

Multistructured models are meant to support the simultaneous expression of several annotation paradigms. For instance, one may want to annotate a text by identifying, independently, its *grammatical* (substantive, adjective, etc.) and its *semantic* (proposition, topic, etc.) structures. To achieve that goal, eAG makes a clear distinction between the representation of **inclusion**¹, which is a modelling relation that makes sense within one annotation paradigm, and **nesting** or **co-occurrence**², which is a fortuitous situation in which two independent elements occur at the same place. And indeed, the eAG syntax for inclusion is *explicit*, while nesting *happens* when two elements X and Y are so that $ref(start(X)) \leq ref(start(Y))$ and $ref(end(Y)) \leq ref(end(X))$ – hence nesting is uniquely defined in terms of reference values.

Yet the notion of chronology is quite impacted by the shift from stand-off to inline markup. In eAG, in order to fit multimedia annotation, several chronologies can be defined, and each node is associated a value from one of those chronologies. In

¹E.g. a *proposition* contains a *topic*.

²A word may happen to be both a *substantive* and the *topic* of a *proposition*: *topic* and *substantive* co-occur; *substantive* is nested in *proposition*; *topic* is included in *proposition*.

a text-only markup setting, a natural chronology is implied by the text itself: the set of inter-character positions. As a consequence, LeAG rests upon that single, natural chronology, that does not even need to be made explicit: tags are simply inserted, within the text stream to be annotated, at the position a corresponding node of an eAG would have made reference to. E.g., annotating the *substantive* in “Let us garlands bring.” is done by inserting a pair of tags as follows: “Let us [Substantive]garlands{Substantive} bring.”

Still, in spite of being considerably simplified compared to eAG, the notion of chronology is still *central* to LeAG, because it is absolutely necessary in order to represent co-occurrence or nesting. Consider the very elementary text stream ABC . A chronology for this text is: $\{start() = before(A), after(A) = before(B), after(B) = before(C), after(C) = end()\}$. Identifying an element Ω between the positions $before(A)$ and $before(C)$ is done as follows: $[\Omega]AB\{\Omega\}C$. The text stream, since it has been added new characters (the ones that constitute the tags), has been altered by this operation.

Yet, interestingly, even in the annotated text stream, the original chronology is still operative to index a very particular text substream, that is, the text stripped from the tags – i.e. the original stream. This may sound tautological; nonetheless, this remark is fundamental, since this bare text stream is the one an editor will consider when she wants to annotate the corpus independently from any previous annotation – that is, when proceeding to multilayer annotation. Indeed, if the editor wants to identify another element ω , ranging from $before(A)$ to $before(C)$, she may insert an opening tag at the position $before(A)$, and a closing tag at $before(C)$, without considering the other tags, resulting in³ $A_1 = [\Omega][\omega]AB\{\Omega\}\{\omega\}C$.

One may also consider that, in the original text stream, $start() = before(A)$ – so the annotation $A_2 = [\omega][\Omega]AB\{\Omega\}\{\omega\}C$ (where $[\omega]$ is inserted at the position $start()$ this time) shall be considered equivalent to A_1 . Similarly, since the two elements Ω and ω are independent, the order in which they are identified shall be indifferent: the two opening (closing, respectively) tags in A_1 and A_2 can be inverted, resulting in two more equivalent markups: $A'_1 = [\omega][\Omega]AB\{\omega\}\{\Omega\}C$ and $A'_2 = [\Omega][\omega]AB\{\omega\}\{\Omega\}C$.

Hence the following relation:

Equivalent LeAG. Let us call **trains of tags** the largest sets of tags, in a LeAG, that are not separated by a character from the original text. Two LeAG are **equivalent** iff they differ only by the order of the tags that belong to their respective trains of tags.

This notion of equivalence actually reflects the fact LeAG, though it is an inline markup syntax, rests upon a notion of chronology that is typical of stand-off markup models. Indeed, one way to interpret the above equivalence relation is by saying that in a LeAG, tags only *make reference* to the position they occupy in the original text stream. Surely, two tags making reference to the same position may be written in any order. Since, in practice, two such tags will not be separated by any character from the bare data and constitute a ‘train of tags’, it follows that in a train of tags, the order in which the tags are written is indifferent.

As a consequence, contrary to XML, the nesting of an element B inside the scope of an element A *cannot* be a means to represent the inclusion of B inside A . Thus a syntax is needed to represent inclusion (cf. 6.2.1). Second, since inserting tags does not alter

³See paragraph 6.2.2 for the actual syntax for multilayer annotation.

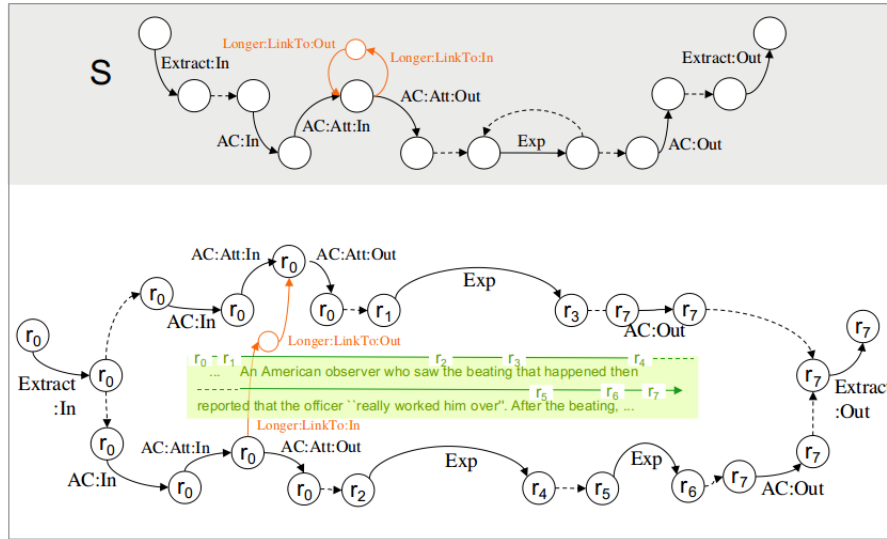


Figure 6.1: A basic anaphoric chain annotation for the extract of *The Village of Ben Suc*, and a corresponding schema.

the chronology that indexes the original text, tags can be considered not to take “any room” along that chronology. This suggests that inserting exogenous resources within the primary resources, e.g. structured comments, can be done *inside special tags that open and close at the same position in the original stream* (cf. 6.2.3).

6.2 The LeAG Syntax

In the following paragraph, based on the above considerations, we gradually introduce the LeAG syntax. The content of the LeAG tags will be defined by means of formulae in which orange characters are constants and italics denotes variables. (Black) parenthesis are mathematical delimiters, not variables or constants. A **field** is either a variable or a formula enclosed in parenthesis. An optional field is followed by the character ?. A field that can be repeated is followed by +, one that is both optional and can be repeated is followed by *. Concatenation is implicit. Space characters are represented by underscores.

6.2.1 Mono-hierarchy of Attributeless Elements

As stated above, some explicit syntax is needed to represent inclusion in a markup model that supports multilayer annotation. This paragraph presents how to express single-layered annotation. The next paragraph extends LeAG towards multilayered annotation.

Elementary spanning elements. An elementary spanning elements (ESE) is the syntactical structure dedicated to the labelling of a section of the primary resources, with the possibility to assess that the current element is included in other elements of the annotation. ESE are represented by a pair of opening and closing tags whose *substance* field has the same value, according to the following:

```
Otag := [ substance ]
Ctag := { substance }
substance := name fathers? ( ,_ ID)?
fathers := _in_ context
```

Above, *name* is the name of the current element and works as a label on the primary resources enclosed by the pair of tags; *context* provides a designation of the elements that contain the current element⁴. The *ID* field will be discussed in the paragraph 6.2.4.

The **content of an element** is constituted of the tags themselves and the whole text (primary resources + tags) they span over.

Rule 6.2.1 The opening and the closing tags defining one ESE cannot belong to the same train of tags.

Back to the example. In order to identify *one* anaphoric chain in the extract of *The Village of Ben Suc*, it suffices to define three element names *Extract*, *AC* and *Exp* for the identification of the extract, the AC and its constituting expressions respectively, and to build the following pairs of opening/closing tags:

- [Extract] and {Extract} ;
- [AC in Extract] and {AC in Extract}, assessing that an AC is included in an extract;
- [Exp in AC] and {Exp in AC}, assessing that an expression is included in an AC.

The following LeAG L_1 annotates the anaphoric chain regarding the young prisoner accordingly.

```
L1 :[Extract]An ARVN officer asked [AC in Extract][Exp in AC]a young prisoner{Exp
in AC} several questions, and when [Exp in AC]he{Exp in AC} failed to answer, beat
[Exp in AC]him{Exp in AC}. An American observer who saw the beating that hap-
pened then reported that the officer “really worked [Exp in AC]him{Exp in AC} over”.
After the beating, [Exp in AC]the prisoner{Exp in AC}[AC in Extract]was forced to
remain standing for hours.{Extract]
```

6.2.2 Grafts: Multilayer Annotation

We now extend the above syntax to multilayer annotation. Multilayer annotation may occur in two distinct situations: first, the schema defines several annotation paradigms; second, one path of the schema is instantiated several times onto the same resources (see Paragraph 5.2.2.4 page 106).

⁴We will see that an element may have more than one father, thanks to the notion of grafts. See paragraph 6.2.2.

The challenge is to make sure that in any case, the tags of a multilayer LeAG shall be unambiguously associated with the layer(s) they are part of. When the set of the elements' names of two co-existing layers do not intersect, assessing to which layer a tag belongs is trivial. At the opposite, simulation-based multilayering, which is prone to self-overlap, will be problematic: in that case, two overlapping elements cannot be discriminated neither on the basis of their name nor by looking at the name of their fathers. Anaphoric chains annotation is a canonical example of such a setting. For instance, in the excerpt of *The Village of Ben Suc*, consider the ACs relative to *the American observer* and *the beating* respectively. A naïve approach making use of the syntax for single-layered annotation would yield the following annotation – which is faulty:

[Extract]An ARVN officer [...] beat him. [AC in Extract]₁ [Exp in AC]₂An American observer who saw [AC in Extract]₃ [Exp in AC]₄the beating{Exp in AC}₅{AC in Extract]₆ that happened then{Exp in AC}₇ reported that the officer “really worked him over”. After [Exp in AC]₈the beating{Exp in AC}₉{AC in Extract]₁₀, the prisoner was forced to remain standing for hours.{Extract]

Indeed, it is undecidable whether the *Exp* element starting at the tag 4 ends at tag 5 or 7. Moreover, there would be no way to ascertain to which *AC* an *Exp* ranging from tag 4 to tag 5 would belong to.

6.2.2.1 Colouring the annotation layers: general strategy

An intuitive disambiguating solution – at least to the human eye – consists in colouring the tags belonging to distinct layers:

[Extract]An ARVN officer [...] beat him. [AC in Extract]₁ [Exp in AC]₂An American observer who saw [AC in Extract]₃ [Exp in AC]₄the beating{Exp in AC}₅{AC in Extract]₆ that happened then{Exp in AC}₇ reported that the officer “really worked him over”. After [Exp in AC]₈the beating{Exp in AC}₉{AC in Extract]₁₀, the prisoner was forced to remain standing for hours.{Extract]

Now it is clear that the element starting at tag 2 ends at tag 5, overlapping with the element starting at tag 4 and ending at tag 7.

Importantly, not only have we coloured differently the elements (2-5) and (4-7) in order to make their respective opening and closing tags match, but also have we given a common colour to the elements (3-10), (4-7) and (8-9), which indicates that the two expressions (4-7) and (8-9) belong to the same AC (3-10), for instance.

Grafts. The notion of *grafts* follows the above intuition. Grafts are coloured LeAGs that are anchored onto an existing LeAG. They express, either locally or at the scale of the whole document, some additional enrichment on top of the annotation that has, at a certain point in time, been done already.

Consider the LeAG L_1 at the end of paragraph 6.2.1. L_1 identifies one AC and its constituting expressions (*Exp*), within an extract. A **graft** must be defined in order to identify, in *the same* extract, another AC, e.g. the AC regarding *the beating*, since this addition will result in a non-hierarchical LeAG. This is done as follows:

1. The element of the existing annotation that will serve as the *context* of the graft is identified: `Extract`, here.
2. A name of ‘colour’, *nameC*, is defined, in the form:

$$nameC := \# \textit{colour}$$

where *colour* is a string that identifies the graft, e.g. “#Red”.

3. The range of the graft is specified by inserting, within the frame of the context element⁵, a pair of colour tags:

$$Otag := [\textit{nameC} \textit{_over_} \textit{context} >$$

$$Ctag := < \textit{nameC} \textit{_over_} \textit{context}]$$

with the *nameC* and *context* fields as defined above. For instance:

```
[Extract]{#Red over Extract> An American observer who saw the beating that
happened then reported that the officer “really worked [Exp in AC]him{Exp in AC}
over”. After the beating [...] <#Red over Extract]{Extract]
```

4. Then *nameC* serves as a context for the top elements of the graft. Here, one AC element spans over the whole graft:

```
[Extract]{#Red over Extract> [...] An American observer who saw [AC in
#Red]the beating that happened then reported that the officer “really worked [Exp in
AC]him{Exp in AC} over”. After the beating{AC in #Red}, [Exp in AC]the prisoner
[...]... <#Red over Extract]{Extract]
```

5. Elements included in the top elements of the graft are defined, their *context* field keeping record of the colour of the upper element. For instance, here, two `Exp` belong to the red AC:

```
[Extract]{#Red over Extract> [...] An American observer who saw [AC in
#Red][Exp in AC#Red]the beating that happened then{Exp in AC#Red} reported
that the officer “really worked [Exp in AC]him{Exp in AC} over”. After [Exp in
AC#Red]the beating{Exp in AC#Red}{AC in #Red}, [Exp in AC]the prisoner [...] ...
<#Red over Extract]{Extract]
```

Similarly, had one `Exp` element had any child, the context field of the tags defining that element would have been `Exp#Red`.

Based on that principle, the LeAG L_2 on Figure 6.3 identifies the three anaphoric chains regarding *the prisoner*, *the beating* and *the American observer* respectively – which is a case of simulation-based multilayer annotation with self-overlap.

⁵For a detailed discussion about how to position the colour tags, see Paragraph 6.2.2.2 just below.

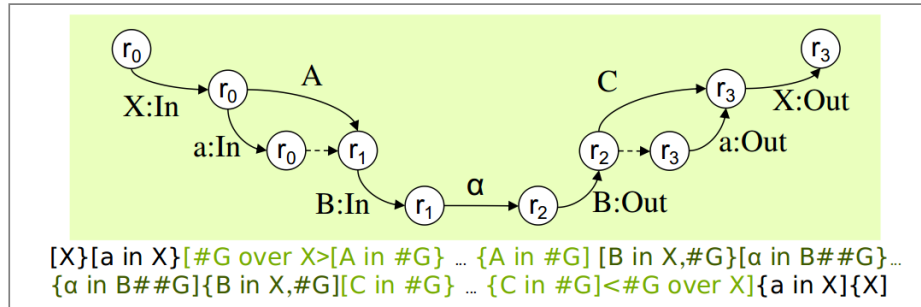


Figure 6.2: A LeAG and a matching eAG, where an element (B) has two fathers, one in the uncoloured hierarchy, and the other in a graft. The colours of B, namely #G and # (uncoloured), are repeated in the context of its son element α .

L_1 : [Extract] [#Blue over Extract] [#Red over Extract] > An ARVN officer asked [Exp in AC] a young prisoner {Exp in AC} questions, and when [Exp in AC] he {Exp in AC} failed to answer, beat [Exp in AC] him {Exp in AC}. [AC in #Blue] {Exp in AC #Blue} An American observer who saw [AC in #Red] {Exp in AC #Red} the beating {Exp in AC #Blue} {AC in #Blue} that happened then {Exp in AC #Red} reported that the officer “really worked [Exp in AC] him {Exp in AC} over”. After [Exp in AC #Red] the beating {Exp in AC #Red} {AC in #Red}, [Exp in AC] the prisoner {Exp in AC} {AC in Extract} was forced to remain standing for hours. <#Red over Extract] <#Blue over Extract] {Extract}

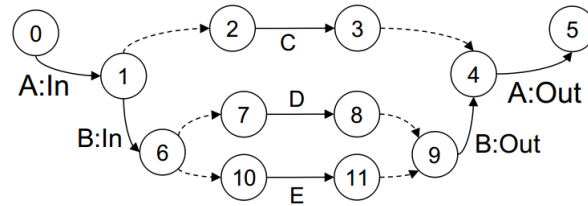
Figure 6.3: Three-layered LeAG.

Complements.

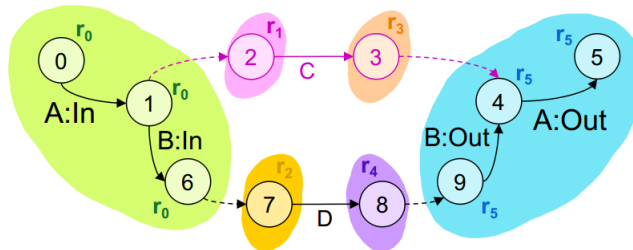
1. Grafts are added on top of an existing annotation spanning over the whole document. Before the first graft is defined, the annotation has to be hierarchical⁶. Thus we can refer to this underlying hierarchical annotation as the **uncoloured hierarchy** of a LeAG. Tags of this hierarchy either have no explicit colour or, when they also belong to a coloured graft, the colour of that graft plus a ‘blank’ colour, # – see element α in Figure 6.2.
2. A graft may be defined either on the underlying hierarchy (Figure 6.2) or on an element from another graft.
3. An element may have several fathers, belonging to grafts or to the uncoloured hierarchy indifferently (cf. element B, Figure 6.2).
4. The span of the graft shall not necessarily equal the one of its context element: see the following paragraph.

6.2.2.2 Positioning Colour Tags: Schema-base and Simulation-based Multilayering

In the above, the indication on how to position the colour tags was voluntarily vague. Indeed, given an underlying hierarchy and a graft on one of the elements of that hierarchy, there are arguably three possible ways of positioning the colour tags ‘within the frame of the context element’. As an example, let us consider the following schema:



Consider the following eAG, validated by the above schema. The coloured areas highlight the nodes that share their reference value and thus, if the eAG was to be represented by a LeAG, shall correspond to the same ToT.



⁶This is not a tough constraint, since a single element spanning over the whole corpus is an elementary hierarchical annotation.

We can deduce from the eAG above that any LeAG expressing the same annotation shall contain six ToT with, for instance, the following ESE tags in them (considering the element C from the eAG belongs to a graft of colour $\#Pink$):

1. ToT₁ contains: $[A]$, $[B \text{ in } A]$
2. ToT₂ contains: $[C \text{ in } \#Pink]$
3. ToT₃ contains: $[D \text{ in } B]$
4. ToT₄ contains: $\{C \text{ in } \#Pink\}$
5. ToT₅ contains: $\{D \text{ in } B\}$
6. ToT₆ contains: $\{B \text{ in } A\}$, $\{A\}$

Then, in theory, the colour tags delimiting the $\#Pink$ could be positioned :

1. Colour tags occur anywhere inside the context element of the graft:


```
[A][B in A] ... [#Pink over A> ... [C in #Pink] ... [D in B] ... {C in #Pink}
... {D in B} ... <#Pink over A] ... {B in A}{A}
```
2. Colour tags occur in the ToT where the first and last tags of the graft occur:

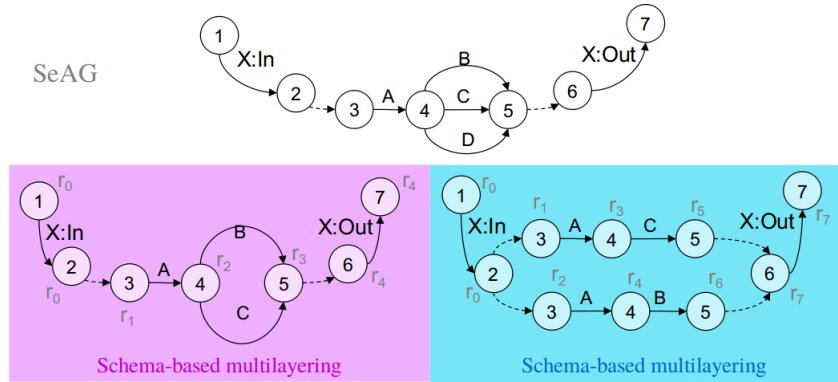

```
[A][B in A] ... [#Pink over A>[C in #Pink] ... [D in B]
... {C in #Pink}<#Pink over A] ... {D in B} ... {B in A}{A}
```
3. Opening colour tags occur in the last ToT containing a tag defining an element of the underlying hierarchy so that this element is common to the annotation layer containing the graft and to the annotation layer containing the context element of the graft – and respectively for closing tags. For instance, in the eAG above, the element A belongs to both the hierarchy “ A contains B which contains D ” (which will be described, in the LeAG, by the underlying hierarchy) and “ A contains C ” (which corresponds to the graft). It is thus reasonable to consider to position the opening colour tag in the same ToT as the opening tag of A , since this is the point from where the two hierarchical annotation layers (underlying and pink) semantically diverge. And indeed, structurally speaking, the end of the edge $A:In$ is the node where the path representing the uncoloured hierarchy and the path representing the coloured hierarchy *diverge* – or, in other words, is the starting node of the two-layered pattern. $[A][B \text{ in } A][\#Pink \text{ over } A> \dots [C \text{ in } \#Pink] \dots [D \text{ in } B] \dots \{C \text{ in } \#Pink\} \dots \{D \text{ in } B\} \dots \{B \text{ in } A\}<\#Pink \text{ over } A][A]$

In the end, it appears that we have three different candidates for positioning the colour tags that define a multilayer annotation. Actually, solution 1 subsumes the other two; but 2 does not subsume 3 and conversely. This means that the two different ways of positioning the colour tags may be made to coexist within a LeAG, in order to enable to differentiate between two different kinds of multilayering: simulation-based and schema-based multilayering respectively.

Let us recall briefly the difference between the two kinds of multilayering. Schema-based multilayering happens when two stacked layers of annotation correspond to two

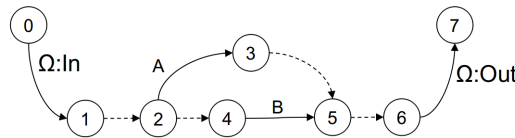
parallel paths of the schema. By contrast, simulation enables to superpose two layers that do not correspond to parallel paths in the schema.

As an illustration, consider the following schema. In that schema, inside of an element X , an element A can be directly followed either by B , C or D . The only part of the schema defining parallel paths is between the nodes 4 and 5. Consider then the two eAG annotations, both identifying two sequences of elements $A - B$ and $A - C$ within one element X , but each according to one of the two kinds of multilayering.

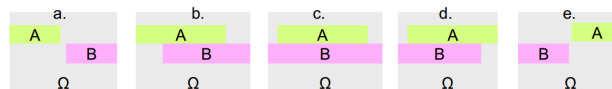


In the schema-based multilayering setting, there is only one element A in the element X , followed by two different elements B and C starting and ending at the same node. In other words, the annotation works as if there was a base layer, for instance made out of the elements A followed by B in X , onto which another layer plugged, without repeating any label of the base layer, and so that the extent of two elements are synchronized. By contrast, in the simulation-based setting, the element X contains the element A twice, on two parallel paths. There is no synchronisation between the elements of each layer, which overlap freely.

Controlling overlap. To justify the necessity of distinguishing, by the syntax of the LeAG document itself, between the two kinds of multilayering, let us take the following, very elementary example of a schema in which two elements overlap but in a specific way.



Simulation-based multilayering enables to express annotations in which any overlapping situation between instances of the elements A and B shall be possible:



To understand the above representation of overlap, for instance, in situation d., the element A will start after a co-occurring element B has started along the annotated content, and end before or after that element also. What this figure shows is that simulation enables to express all kinds of overlap between A and B .

Interestingly, by contrast, the only overlapping situations that are allowed in a **schema-based** multilayer setting are the situations a. and b., i.e. the situations in which A does not start after B started, and B does not end before A has ended. In other words, SeAG schemas can control the way elements overlap **in schema-based multilayering settings**.

Obviously, if there is no differential syntax for the two kinds of multilayering, it is not possible to benefit from this aspect of SeAG schemas. On the contrary, by making so that the default, or more natural syntax for grafts be the one for schema-based multilayering, we can make sure that all the grafts defined according to that syntax be controlled by the schema; those explicitly written in the syntax for simulation-based multilayering will not, but in a conscious way.

Positioning colour tags: the two kinds of multilayering. Grafts in LeAG documents will be parsed against a schema accordingly with the following convention⁷

1. Schema-based layering is expressed by positioning the opening (resp. closing) colour tag inside the same ToT as the first (resp. last) tag that makes use of that colour in its context field.
2. Simulation-based layering is expressed positioning the opening (resp. closing) colour tag inside the ToT that occurs at the position where the bifurcation (resp. convergence) with the underlying hierarchy occurs in the corresponding eAG.

Example. Consider the eAG/SeAG couples represented on page 136. The eAG on the left, which illustrates schema-based multilayering, can be written in LeAG as follows:

```
[X] ... [A in X] ... {A in X} [#Pink over X]>[B in X][C in #Pink]
... {B in X}{C in #Pink}<#Pink over X}... {X}
```

By contrast, the eAG on the right, which is an example of simulation-based multilayering, may translate into LeAG as follows:

```
[X] [#Pink over X] ... [A in X] ... [A in #Pink] ... {A in X}[B in X]
... {A in #Pink}[C in #Pink] ... {B in X} ... {C in #Pink}... <#Pink over X>{X}
```

6.2.3 Standard Inserts: Attributes, Structured Comment

So far, we have seen how to label the primary resources by means of entangled hierarchies of elementary spanning elements. Still, editing is not only about labelling: sometimes, additional, structured information must be added on top of the labels. In XML, this kind of information constitutes elements' attributes; still, adding attributes

⁷Actually, this convention is not arbitrary, but is based on algorithmic considerations. See page 180.

to an element is like annotating the element itself, that is, for the editor, inserting secondary, structured data that does not bear on the primary resources but on the tags. Similarly, providing the editor with means to express critical information, not by labelling the primary resources, but by inserting assertions is a useful feature. Introductions, comments and punctual notes, in their digital form, fall into that category of annotations.

Attributes and punctual comments share the property of not being expressible with elementary spanning elements. In LeAG, both will be represented by means of **inserts**. An insert is similar to void elements in XML in that: (1) it is both opening and closing, which means, in the LeAG vocabulary, that inserts start and end at the same position; (2) it is self-contained, in the sense that the tag representing the insert is the insert's content.

Attribute insert: general syntax. The syntax of an attribute insert respects the following formula:

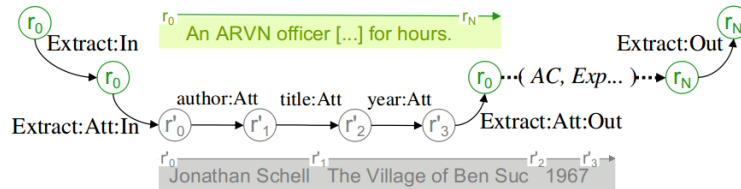
$$\text{Insert}A := [\text{Att_of_ context _ ;_ LeAG}]$$

where *context* is the coloured name of the element whose attributes are described in the insert, and *LeAG* is some a LeAG annotation stripped from its containing element, and constitutes the content of the attributes.

Attribute insert: example. So far, the passage of *The Village of Ben Suc* as a whole was simply labelled as an **Extract**. The following LeAG provides, as attributes of the **Extract**, the author's name, the title and the publication year of the novel:

```
[Extract]{Att of Extract ; [author]Jonathan Schell {author}[title]The
Village of Ben Suc{title}[year]1967 {year}}An ARVN officer, [...] for
hours.{Extract}
```

The insert corresponds, in eAG, to a hierarchy of elements bearing the suffix **:Att**, included in the element **Extract**:



In the LeAG, there is no need neither to specify the **:Att** suffix for the elements defined inside the attribute insert, nor to indicate in the context field of the top elements among them, that they are included in the insert⁸. The same applies to comment inserts:

Comment inserts: general⁹ syntax. The general syntax of a comment insert is the following:

$$\text{Insert}C := [\text{name _in_ context (,_ ID) ? _ ;_ LeAG}]$$

where *name* is the name of the insert, *context* is the coloured name of the elements the insert is the son of and *LeAG* structured data conform to the LeAG model, constituting the content of the comment. The *ID* field will be discussed in Paragraph 6.2.4.

⁸*Id est*, there is no need to write `[author in Att of Extract]`, for instance.

⁹A refinement of the following syntax will be proposed in the paragraph 6.2.4.

Comment insert: an example. The following LeAG incorporates a *comment* regarding the context of *The Village of Ben Suc*:

```
[Extract]An ARVN[Comment in Extract ; [Att of Comment ;
[authorOfComment]Barrellon et al.{authorOfComment}]The mention of the
[acronym]ARVN{acronym} refers to the Vietnam War.] officer asked [...] for
hours.[Extract]
```

A comment being an element, it may possess attributes, as illustrated above (e.g. to specify the name of its authors).

Inserts in a train of tags. The case of inserts within a train of tags has to be discussed. Consider the LeAG $[A] \dots \{A\} [B \text{ in } A ; LeAG] [A] \dots \{A\}$. In the absence of a schema, it is not possible to assess to which A element B belongs. If there is a schema that does not restrict the position of the element B either at the beginning or at the end of the element A , neither.

Second, consider $[A] \dots [B \text{ in } A ; L_1] [C \text{ in } A ; L_2] \dots \{A\}$. The LeAG itself is not ambiguous: it states that the inserts B and C occur at the same position. Yet, in the perspective of parsing the LeAG into an eAG, since in the corresponding eAG, two inserts will form a sequence, there is no indication in the LeAG about which insert will come first. The following conventional rule clarifies those situations:

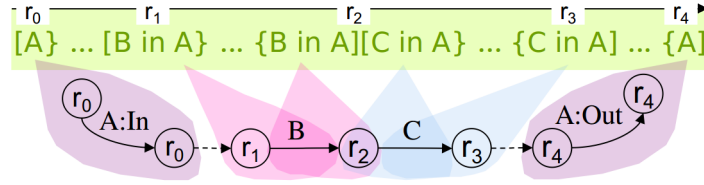
Rule 6.2.3 When there is no schema or when the schema does not clarify the following situations, it shall be considered that (1) when an insert occurs in a train of tags where an opening and a closing tags identically match the context field of the insert, then the insert conventionally belongs to the *opening* element; (2) when two inserts with the same context field occur in the same train of tags, the order in which the tags appear along the text document provides a conventional order between the inserts.

6.2.4 Links and Quoting Elements

The last aspect of eAGs that needs to be translated into LeAG is links or quotes. We have seen that in eAG, links and quotes are expressed harmoniously with the other elements (i.e. by means of nodes and edges) and, for that reason, can be properly *validated*. In particular, compared to XML where a link is but an ID/IDREF pair, in SeAG/eAG, the nature of the two elements connected by a link can inherently be restricted. Still, since links and quotes denote distant connections across the corpus that may result in cyclic annotations (i.e. along the text stream, the beginning of an element comes after its end), it is not possible to represent them by means of pairs of tags along the text stream. Thus, Linear extended Annotation Graphs make use of an additional feature: the ID field.

ID fields work as an identifier of either the source or the end of a connection (link/quote), hence enabling to position the extreme *nodes* of such elements inside the LeAG, that is, to position the elements themselves. Yet, ID fields are not *tag* identifiers. Indeed, regardless of the parsing strategy adopted, there is no one-to-one correspondence between the *tags* of a LeAG and the *nodes* of an eAG expressing the same annotation, as evidenced below¹⁰:

¹⁰Coloured shapes relate the eAG nodes/edges to the tags that set their position/label.



Indeed, because the element A contains other elements, the tag $[A]$ translates into *two* nodes whose reference values point towards the position of $[A]$ inside the document, connected by an edge $A:In$, while the tag $[B]$ relates to *one* node only. Conversely, two tags may relate to the same node: since the element C starts where B ends, both $\{B\}$ and $[C]$ relate to the node that separates B and C .

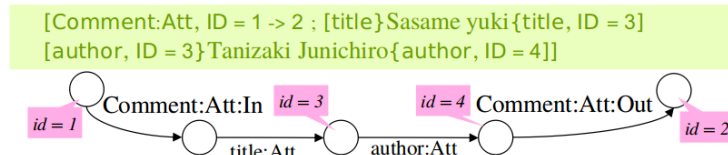
Yet, a finer correspondence between the LeAG tags and a *subset* of the nodes of the corresponding eAG can be exploited for expressing links and quotes: (1) an opening tag positions (and hence, matches) the root of the corresponding element in the eAG; (2) a closing tag positions the leaf of the corresponding element in the eAG; (3) an insert positions both the root and the leaf of the corresponding element in the eAG. ID fields exploit that connection, as follows.

ID fields. Since opening and closing tags of ESE relate to either the root or the leaf of an element in the corresponding eAG, ESE *ID* fields contain a singleton value K . *A contrario*, an insert *ID* shall possibly designate the root and the leaf of the corresponding eAG element and thus contains a pair of values M and N :

$$ID := ID_ _ K \text{ (singleton syntax)}$$

$$ID := ID_ _ M _ _ _ N \text{ (pair syntax)}$$

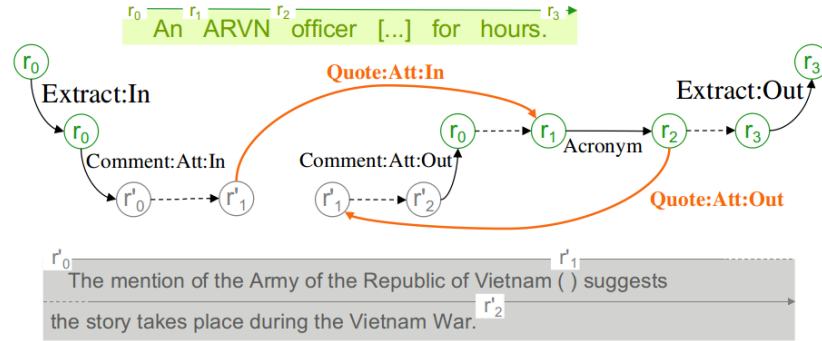
Basic example. Let us consider the following comment and a matching eAG (pink flags represent the node identifiers):



Noteworthy, the relation between ID values and root/leaves nodes is only *surjective*. Thus, the ID of the closing tag of an element and that of an element that immediately follows have to be equal (e.g. $\{title, ID = 3\}$ and $[author, ID = 3]$, above).

The syntax for links and quotes are based on that mechanism – plus some improvements on the insert syntax.

Quote elements. Quote elements enable to include an element identified in the primary resources within a comment. In eAG, quoting the ARVN acronym from the extract of *The Village of Ben Suc* within a comment can be done as follows:



The two (orange) edges permit to *structurally include* the quoted element inside the comment element.

In LeAG, since the content of a comment has to be written inside the insert itself, quoting, inside the *LeAG* field of a comment, an element that has been identified elsewhere in the annotation cannot be done but by reference. Therefore, quoting elements appear as special comment inserts, whose *LeAG* field has been replaced by an *ID* field (with the pair syntax):

$$\text{Quote} := [\text{name}(\text{_in_context})?(\text{_ ,_ ID}_1)? \text{_ ;_ ID}_2]$$

$$\text{ID}_i, i \in \{1, 2\} := \text{ID}_=_ M_i \text{_ ->_ } N_i$$

The pair of values of the ID_2 field must then refer to some tag(s) somewhere else in the LeAG that delimit either an element or a sequence of elements.

Quote: example The LeAG representing the above eAG is:

```
[Extract][Comment in Extract ; The mention of the Army of the Republic of
Vietnam ([Quote ; ID = 1-> 2]) refers to the Vietnam War.]An [Acronym in
Extract, ID = 1}ARVN{Acronym in Extract, ID = 2} officer [...] hours.{Extract]
```

This LeAG does correspond to the eAG above, since it states that the *Extract* contains a *Comment:Att*, made out of some not annotated text (which translates into an epsilon edge), followed by a *Quote* containing an annotation graph whose root and leaf have the identifiers '1' and '2' respectively; *Extract* further contains an *Acronym*, whose root and leaf identifiers are '1' and '2' respectively.

Links. An eAG link is an element whose root is a node from an element and whose leaf is a node from another element.

First, to represent such a graph in LeAG, we need to be able to identify a node *inside* any element. Consider the link in Figure 6.1. It connects the internal nodes of two *AC:Att* elements that contain nothing but those nodes. Yet, the *ID* field of an insert with no *LeAG* field, suit to represent those *AC:Att* elements, only identifies the root and leaf of the matching element, not an internal node. To fill this gap, we define **void inserts**:

$$\text{VoidInsert} := [\text{in_context} \text{_ ,_ ID}]$$

$$\text{ID} := \text{ID}_=_ N$$

Such an insert neither has a name nor a *LeAG* field, but it does have a context (the element it is included in) and an *ID* field. Placed immediately after an opening tag, e.g. [A], a void insert [in A, ID = 1] enables to give the identifier '1' to a node that, in the corresponding eAG, is the node ending the *A:In* edge.

Second, we need a means to express that an element may start inside an element and end inside another one. For such a special element, we defined **link insert**:

```
Link := [ name :LinkTo_in_ context ,_ ID( ;_ LeAG)? ]
ID := ID=_ M _->_ N OR ID=_->_ N
```

The *LeAG* field defines the content of the link; if empty, the link is an edge. The leaf of the link, identified by the value of the variable *N* above, must be an internal node of some element, represented elsewhere by a void insert.

Link: example. Figure 6.1 illustrates how to annotate different, overlapping AC in an extract, and how links could reify an order relation between them. The following LeAG expresses the same annotation, extended to three ACs (*the prisoner, the beating, the American observer*) as in the eAG on Figure 6.1:

```
[Extract]An ARVN officer asked [AC in Extract][Att of AC ;
[Longer:LinkTo, ID = -> 2][Longer:LinkTo, ID = -> 1]][Exp in AC]a
young prisoner{Exp in AC} questions, and when [Exp in AC]he{Exp in AC}
failed to answer, beat [Exp in AC]him{Exp in AC}. [#Blue over Extract]>[AC
in #Blue][Att of AC#Blue ; [in Att#Blue, ID = 1]][Exp in AC#Blue]An
American observer who saw [#Red over Extract]>[AC in #Red][Att of AC#Red
; [in Att#Red, ID = 2][Longer:LinkTo, ID = -> 1]][Exp in AC#Red]the
beating<#Blue over Extract>{Exp in AC#Blue}[AC in #Blue] that hap-
pened then{Exp in AC#Red} reported that the officer “really worked [Exp
in AC]him{Exp in AC} over”. After [Exp in AC#Red]the beating{Exp in
AC#Red}[AC in #Red]<#Red over Extract>, [Exp in AC]the prisoner{Exp in
AC}[AC in Extract]was forced to remain standing for hours.{Extract}
```

6.3 Summary and Notation

The following table aims at summarizing the LeAG syntax that has been introduced in the previous paragraph.

Tags. LeAG has an inline markup syntax. An annotation graph is made out of tags. A tag is a sequence of characters delimited by a starting character *St* and an ending character *Ed*. The characters in between belong to the tag, whose structure is the following :

$$\text{Tag} := \quad St \text{ substance } Ed$$

The typology of LeAG tags is given in Table 6.1.

Notation. Based on the above table, we will from now on be able to speak of the *name, context, colour*, etc. fields of a tag, or that of an element or a colour – if appropriate¹¹.

¹¹The definitions of elements and colours are given below. Spanning elements and colours being made out of two tags, one can speak of a field for the whole element only for those fields that are identical for the two tags, e.g. the *name* field. On the contrary, the two delimiting tags of a spanning element do not have the same *ID* field, so talking about the *ID* field of an element does not make any sense.

Tag type	St char.	Ed char.	substance field formula	substance field description
opening tag	[]	$name\ father? (_ ID)?$	<ul style="list-style-type: none"> • $name \in \mathcal{L}$ • $father := _in_ context$ <ul style="list-style-type: none"> - $context := radical (_ radical)^*$ - $radical := nameF? colour^*$ - $nameF \in \mathcal{L}$ - $colour := \# cn$ - cn is an alphabetical string • $ID := ID = N$ <ul style="list-style-type: none"> - $N \in \mathbb{N}$
closing tag	{	}	same as opening tags	same as opening tags
void insert	[]	$in_ context _ ID$	<ul style="list-style-type: none"> • $context$: same as opening tags • ID : same as opening tags
comment insert	[]	$name_in_ context (_ ID)? _ LeAG$	<ul style="list-style-type: none"> • $name$: same as opening tags • $context$: same as opening tags • $ID := ID = M \rightarrow N$ <ul style="list-style-type: none"> - $N, M \in \mathbb{N}$ • $LeAG$ corresponds to an independent well-formed LeAG, stripped from its containing element. By independent, we mean that the name of the current insert needs not be specified as the context of the top-level elements of the independent LeAG.
attribute insert	[]	$:Att_of_ radical (_ ID)? _ LeAG$	<ul style="list-style-type: none"> • $name$: same as opening tags • $radical$: same as opening tags • ID := same as comment inserts • $LeAG$: same as comment inserts
link insert	[]	$name :LinkTo_in_ radical _ ID (_ LeAG)?$	<ul style="list-style-type: none"> • $name$: same as opening tags • $radical$: same as opening tags • $ID := ID = M? \rightarrow N$ <ul style="list-style-type: none"> - $N, M \in \mathbb{N}$ • $LeAG$: same as comment inserts
quote insert	[]	$name_in_ context (_ ID_1)? _ ID_2$	<ul style="list-style-type: none"> • $name$: same as opening tags • $context$: same as opening tags • ID_1, ID_2 : same as comment inserts
opening colour tag	[>	$name_over_ radical$	<ul style="list-style-type: none"> • $name$: same as $colour$ in opening tags • $radical$: same as opening tags
closing colour tag	<]	same as opening colours	same as opening colours

Table 6.1: Summary of the syntax for LeAG tags.

Given an element (or colour) A , the set of x fields of A will be denoted $A.x$. For instance, considering a tag $A := [\text{Line in Poem, Stance\#R, Chorus\#B}]$:

1. $A.name = \{\text{Line}\}$. Noteworthy, since the *name* field cannot be multiple in any tag, $A.name$ will always be a singleton and will conveniently be used as a value.
2. $A.radical = \{\text{Poem ; Stance\#R ; Chorus\#B}\}$.

The concatenation of the objects composing a set¹² $aSet$ will be denoted \prod_{aSet} . For instance here, $\prod_{A.colour} = \#R\#B$.

In case of a composite field f (e.g. *father*), the above notation may be used to denote a particular subfield of f (e.g. $f.nameF$).

The concatenation of two values will be either implicit (no operator) or, when clarity demands so, denoted by the character \cdot : for example, the concatenation of the name of the element A and of the concatenation of its colours will be written $A.name A.colour = A.name \cdot \prod_{A.colour}$ – and will equal Poem\#R\#B .

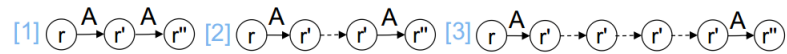
¹²Note that, strictly speaking, there is not *one* way of concatenating the objects of a set (any ordering of the objects yields a different value); yet, in the following, the order of the objects in the concatenation will always be of no importance when that concatenation operator will be used.

Chapter 7

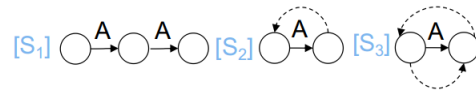
An Efficient Parser for Linear Extended Annotation Graphs

Let us consider that two eAG are **isomorphic** iff there is a bijective morphism ϕ between them so that a node and its image by ϕ share the same reference value.

LeAG is designed as a markup syntax for eAG. Ideally, there should have been a bijection between LeAGs and the classes of isomorphic eAGs. Yet, this is not the case: first, because two equivalent LeAG documents shall translate into the same eAG, and second, because several non-isomorphic eAGs could match a given LeAG – which is clearly problematic when considering parsing LeAG documents into eAG. For instance, the elementary LeAG $[A] \dots \{A\} [A] \dots \{A\}$ may reasonably translate into either of the following:



or any eAG made out of a sequence of two edges labelled A with the right reference values, separated by any number of epsilon edges. The problem is we cannot, in the absolute, prefer one eAG over the others, since all of them do represent the fact the LeAG document contains two A elements in a row – and also, and most importantly, because the different eAG *will not be validated against the same schemas*. Indeed, considering the three SeAGs below, the above eAG [1] is validated by the schema $[S_1]$ only, [2] by both $[S_2]$ and $[S_3]$, and [3] by $[S_3]$ only.



Choosing one solution against the others thus cannot be done but by considering a predefined schema. Hence parsing a LeAG means: given a SeAG, yielding a valid eAG that ‘represents well’ the LeAG – if such an eAG exists.

In the following, we discuss how to design a deterministic schema-aware LeAG parser. The whole discussion that follows is ‘up to isomorphism’. First, we introduce some elements of notation, that will be useful for the following paragraphs. Then we consider

the problem of parsing LeAG documents against a schema and show that focussing on a certain family of LeAG-SeAG couples is beneficial in terms of algorithmic complexity. Eventually, we define a single pass, SeAG-aware LeAG parser.

7.1 General Parsing Strategy

The parsing setting we consider here is the following: given a SeAG S , a LeAG L , we want to design a parser that deterministically yields an eAG I_S that is validated by S and that matches L , i.e., so that there be a bijective function Φ between the set \mathcal{E}_K of the elements of L and the set \mathcal{E}_{I_S} of the elements of I_S , so that the following conditions are verified:

1. $\forall e \in \mathcal{E}_K$, if e is an ESE not enclosed within an insert, or a link, the name of $\Phi(e)$ is $e.name$; else, the name of $\Phi(e)$ is $e.name:Att$;
2. $\forall e \in \mathcal{E}_K$, $ref(root(e)) = ref(root(\Phi(e))) \wedge ref(leaf(e)) = ref(leaf(\Phi(e)))$;
3. $\forall e \in \mathcal{E}_K$, if the root, leaf or first node of e is given the identifier value N by means of the appropriate ID field, the corresponding node of $\Phi(e)$ has N as an identifier value;
4. $\forall e, f \in \mathcal{E}_K$ so that e is the preceding element of f along some annotation layer of L , then $\Phi(e)$ precedes $\Phi(f)$ in I_S ;
5. $\forall e, f \in \mathcal{E}_K$ so that f is included in e in L , then $\Phi(f)$ is included in $\Phi(e)$ in I_S .

At this point, the reason why we need a parser is because this, indirectly, provides the LeAG markup syntax with a validation mechanism. This means that, if there is an eAG that verifies the above conditions and that is validated by the schema S , then, the output of the parsing algorithm with L and S as parameters, cannot be void. Conversely, it means that if the output of the parsing algorithm, with L and S as parameters is not void, then, L can be regarded as a valid LeAG on behalf of the schema S .

Noteworthy, giving validation up to a parsing algorithm is coherent with the fact the eAG/SeAG model benefits from a matrix representation model that enables validation to be guaranteed by construction, that is, assessed in the very course of the manufacture of an eAG, given a predefined schema. By defining data structures that mimic the aforementioned matrix representation – or benefit from its properties – to represent the schema and the eAG, as an output of the parsing of a LeAG L , then validation can indeed be checked by parsing the LeAG, since parsing means building an eAG in the template of a schema: if this construction is possible, the LeAG is valid, and is not if not.

Performance-wise, what matters, regarding the parsing/validating algorithm, is time complexity mainly [126]. The goal is to design an algorithm with a linear theoretical time complexity, in terms of the size of the document. This must be compared to the complexity of OWL, RDFS, ScheX, SHACL, that are the only validation mechanisms that handle cyclic data.

The following paragraph introduces the general parsing strategy we adopted. Some restrictions on the nature of schemas and LeAG documents were taken in order to manage acceptable parsing costs: we introduce and justify those. In the last paragraph, we provide a synthetic view of the parsing algorithm and quantify its theoretical time complexity.

7.2 Parsing Strategy: Elements of Design

The parsing algorithm we designed works in a single pass, and its theoretical time complexity is linear in terms of the documents' number of tags (see Paragraph 7.3.3). The main steps of the algorithm are the following:

1. Initiate the data structures – see 7.3.1.
2. For each ToT:
 - (a) Consider the colour tags. Register the colours that are newly declared in the ToT, if they are used in a tag of the ToT or not. Register the colours that end in the ToT.
Design discussion: page 172.
 - (b) Generate the (possibly disconnected) subgraphs that correspond to the ESE defining tags only. This graph may contain pending nodes (incomplete edges). Register to which hierarchical level the roots and leaves of each subgraph belong.
Design discussion: page 148.
 - (c) Build the disconnected linear subgraphs that correspond to the different sets of inserts sharing the same context. Connect their root and leaf to the nodes of the ESE subgraphs marked as belonging to the same level appropriately.
Design discussion: page 157.
 - (d) Connect the remaining disconnected subgraphs relative to the ToT by sequences of ϵ edges as allowed by the schema. Those sequences of ϵ edges must connect nodes belonging to the same hierarchical level appropriately.
Design discussion: page 157.
 - (e) Connect the first node of each hierarchy of elements to the last node of the same hierarchy belonging to a previous ToT, by means of sequences of ϵ edges as allowed by the schema. This operation must result in a bigger (incomplete – unless the ToT is the last of the LeAG) eAG containing all the nodes and edges related to the ToT. The pending nodes whose reference values correspond to the current ToT remain for being connected to a later ToT.
Design discussion: page 157.

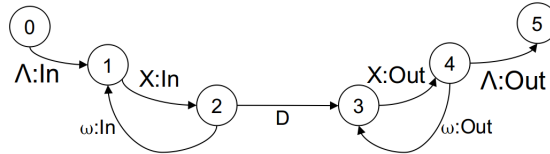
Yet, this strategy imposes some restrictions on the nature of LeAG document / SeAG schema pairs, as defined below.

7.2.1 Restrictions on ESE Defining Tags

SeAG/LeAG couples must verify the following conditions:

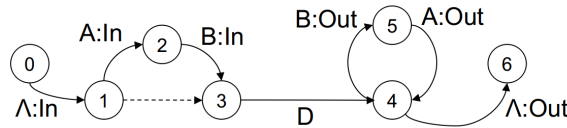
- (C1) No two subgraphs of the schema define two elements with the same name.
- (C2) Any subgraph of the schema that contains ESE defining edges and that forms an Hamiltonian path can be instantiated into a hierarchy of elements.

First, (C2) implies in particular that the schema contains no recursive element's definition. Indeed, recursive elements' definition imply that there is a cycle in the schema that leads back to the root of an element from inside this element, this, before the ending edge of that element is encountered – so the opening edges of those elements are included in the cycle while the ending edge is not, which implies there is no way to unfold the sequence of edges that the cycle contains into a hierarchical pattern. It is important to ban recursive definitions in the perspective of parsing LeAGs, since this ban prevents from the situation in which an element named X be included in another element named X , both starting at the same reference point, which is ambiguous in the LeAG notation. For instance, the following schema defines the elements X and ω recursively:

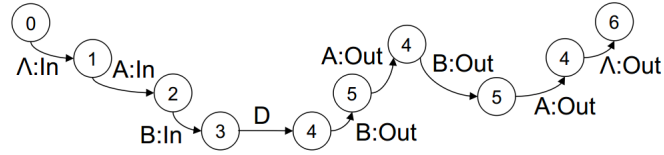


Consider the following ToT: $[\Lambda]\{X \text{ in } \Lambda\}[\omega \text{ in } X, \text{ID} = 1]\{X \text{ in } \omega\}[\omega \text{ in } X, \text{ID} = 2]\{X \text{ in } \omega\}[D]$. It describes the start of a hierarchy of elements that is allowed by the schema, as follows : $\Lambda > X > \omega > X > \omega > X > D$. Yet there is no way to assess whether the tag named X , bearing the ID value 1, is the ancestor of the tag named X , bearing the ID value 2, or the other way round. Avoiding recursive definitions in the schema prevents from such ambiguities.

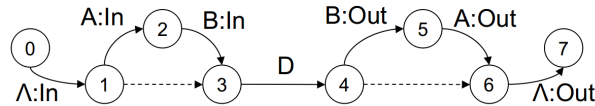
Second, (C2) prevents from writing schemas so that any graph instantiating twice, in a row, the sequence of edges corresponding to a cycle from the schema, be not well-formed. Consider the following SeAG:



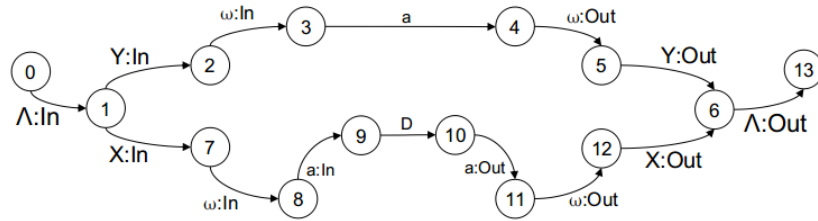
The schema defines an element D that is either directly included in the top element Λ , or included in B , that is included in A , itself included in Λ . Note that the elements' definitions are not recursive here. Defining hierarchies of elements in which some hierarchical steps are optional is a key feature of SeAG; yet, the way it is done here, by recurring to a cycle made out of $:Out$ edges, somehow hijacks the purpose of cycles in SeAG, that is, to define repeatable sequences of edges. Indeed, no graph instantiating the cycle of the schema twice in a row, as follows, can be well-formed:



Still, (C2) does not refrain from expressing hierarchies of elements with optional hierarchical steps. The above SeAG can be reformulated in order to respect (C2) as follows:



In this context, (C1) guarantees that each tag name be associated with at most one triple $LTT' \in \mathcal{L} \times \mathcal{T} \times \mathcal{T}$ from the schema that is considered for the parsing process. This implies that, whenever the parser reads a given tag, regardless of the context of the element it defines in the LeAG, then a unique edge from the schema can be associated to that tag. It is quite clear that schemas in which two elements can have the same name do not provide that interesting property. See for example the following SeAG :



A tag $[a \text{ in } \omega]$ occurring in a LeAG could then be associated with either the edge $(a, 3, 4)$ or $(a : In, 8, 9)$ from that schema. Choosing amongst those two shall then demand to take the context of the element a of the LeAG into account – which would mean, in this elementary example, assessing whether the element a has X or Y as an ancestor.

From this point of view, (C1) helps designing a parser able to manage trains of tags in which the opening and closing tags appear in a random order within a train of tags, since it is not necessary to take the context of a tag to determine the edge of the schema that matches the tag.

We have seen that since (C2) bans recursive elements, that are not well expressed in LeAG, as well as cycles in the schema that hijack cyclic patterns from their intended use: for those two reasons, the condition seems relevant. Yet (C2) also provides an interesting property to the eAGs that will derive from a LeAG, and that can be exploited also in order to design a LeAG parser that, at least for LeAGs made out of ESE only, is order agnostic in terms of the tags within ToTs.

Notation. Be G a SeAG (or an eAG), X an element from G containing other elements. Then let us denote $first(X)$ the end of the $:In$ edge of X and $last(X)$ the summit of its $:Out$ edge.

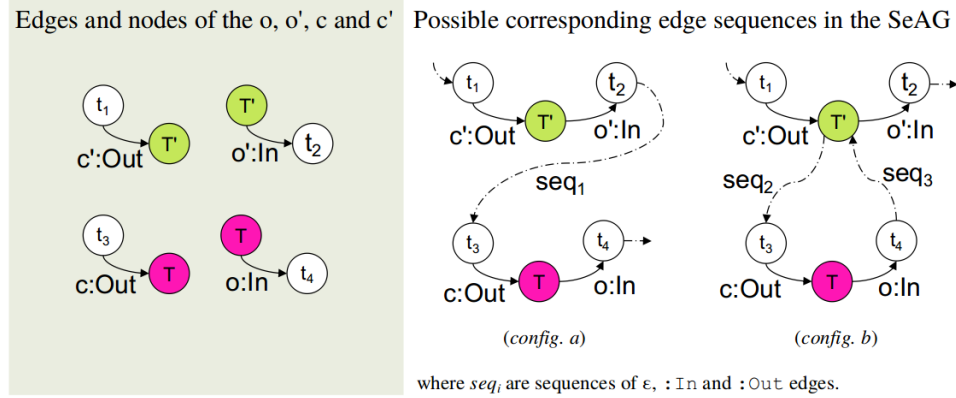
Property 7.2.1. Be a SeAG S , verifying conditions (C1-2). Be an eAG E validated by S . Then: be $\mathcal{C}_{r,H}$ a set of elements from E belonging to the same hierarchy H and ending at the reference point r ; be $\mathcal{O}_{r,H}$ a set of elements from E belonging to H and starting at r .

1. $\forall o_1, o_2 \in \mathcal{O}_{r,H}, type(root(o_1)) \neq type(root(o_2))$;
2. $\forall c_1, c_2 \in \mathcal{C}_{r,H}, type(leaf(c_1)) \neq type(leaf(c_2))$;
3. $\forall (o_1, o_2), (o'_1, o'_2) \in \mathcal{O}_{r,H}^2, type(first(o_1)) = type(root(o_2)) = type(first(o'_1)) = type(root(o'_2)) \Rightarrow first(o_1) = root(o_2) \wedge o_1 = o'_1 \wedge o_2 = o'_2$
4. $\forall (c_1, c_2), (c'_1, c'_2) \in \mathcal{C}_{r,H}^2, type(leaf(c_1)) = type(last(c_2)) = type(leaf(c'_1)) = type(last(c'_2)) \Rightarrow leaf(c_1) = last(c_2) \wedge c_1 = c'_1 \wedge c_2 = c'_2$
5. $\forall (c, o) \in \mathcal{C}_{r,H} \times \mathcal{O}_{r,H}, type(last(c)) \neq type(first(o))$
6. $\forall (c, o), (c', o') \in \mathcal{C}_{r,H} \times \mathcal{O}_{r,H}, type(leaf(c)) = type(root(o)) \wedge type(leaf(c')) = type(root(o')) \Rightarrow o = o' \wedge c = c'$

Proof. 1. to 5. Hierarchical structures in an eAG are linear. In an eAG made out of ESE only, if several elements from the same hierarchy start (resp. end) at the same reference point, then their $:In$ (resp. $:Out$) edges are consecutive (i.e. separated by $:In$ (resp. $:Out$) edges or ϵ edges). Thus, the instance contains a linear sequence of $:In$ (resp. $:Out$) edges mixed with ϵ edges. Suppose two edges from that sequence share the same type. A common result is that a linear sequence from an eAG repeating a type value instantiates a cycle from the schema. Yet (C2) implies that there is no Hamiltonian subgraph of the schema that is constituted of edges suffixed $:In$ (resp. $:Out$) only, or of such edges mixed with ϵ edges only.

6. Suppose there were two pairs $(c, o), (c', o') \in \mathcal{C}_{r,H} \times \mathcal{O}_{r,H}$, so that $type(leaf(c)) = type(root(o)) \wedge type(leaf(c')) = type(root(o'))$ and $o \neq o' \vee c \neq c'$ [H]. First, if $o \neq o'$, for instance, then we know that $type(root(o)) \neq type(root(o'))$, which also implies that $type(leaf(c)) \neq type(leaf(c'))$. Since $c \neq c'$ implies the same inequalities over node types, we can infer that in any case, [H] $\Rightarrow type(leaf(c)) = type(root(o)) \wedge type(root(o)) \neq type(leaf(o')) \wedge type(root(o')) = type(root(c'))$. Second, let $e:Out$ denote the ending edge of an element e and $e:In$ its opening edge. Since o, o', c and c' all belong to the same hierarchy, then there is a linear subgraph of E to which $c:Out, c':Out, o:In, o':In$ belong. This means that there is a subgraph of the schema whose label sequence respects either the following configuration a of b¹:

¹ o and o' on the one hand, and c and c' on the other hand, could be inverted. This changes nothing to the proof.



The seq_i subgraph represents the edges that are instantiated in E in order to connect the four edges ($c:Out$, $c':Out$, $o:In$, $o':In$) we know belong to a linear sequence whose nodes all share the same reference value. *Config. a*. If there is an edge labelled $o':Out$ in seq_1 or if $name(o') = name(c)$, then the element o' in E begins and ends at the same position, which is not possible since it is an ESE. Then no edge of seq_1 is labelled $o':Out$. It follows that along the linear subgraph of the schema in *config. a*. depicted above, an edge labelled $c:Out$ is encountered before an edge $o':Out$ is. Since the instantiation of seq_1 in E is hierarchical, it means that there is an edge labelled $c:In$ in seq_1 . Yet this would imply that the element c also opens and closes at the same reference point, which is not possible since c is an ESE. So *config. a*. is not possible. *Config. b*. For the same reasons as for *config. a*., we can exclude that seq_2 contains anything but $:Out$ or ϵ edges, and that seq_3 contains anything but $:In$ or ϵ edges. Then the only solution for the Hamiltonian subgraph represented in *config. b*. to respect condition (C2) is that all the $:Out$ edges in seq_2 have a corresponding $:In$ edge in seq_3 , in the right order as to form a hierarchical sequence, and that $name(c) = name(o)$. Yet this, in particular, implies that there is an ESE named $name(c) = name(o)$ in E that opens and ends at the same reference point, which is not possible. \square

Corollary. Be a LeAG L , two ESE-defining tags t_1 and t_2 belonging to the same ToT of L and so that t_1 and t_2 define two elements belonging to the same hierarchy². Be $L_1T_1T'_1$ and $L_2T_2T'_2$ the triples associated with each tag. Then $T'_1 = T_2$ implies that the node whose type equals T'_1 associated with the tag t_1 (i.e. the leaf of the corresp. element if t_1 is a closing tag, or the first node of the corresp. element) is the same node as the node of type T_2 associated to the tag t_2 (i.e. the root of the corresp. element if t_2 is a closing tag, or else the last node of the corresp. element).

Definition 7.2.1 : tags belonging to the same hierarchy. Be L a LeAG, X a ToT of L , t_1 and t_2 two tags belonging to X . Then t_1 and t_2 define elements belonging to the same hierarchy iff one of the following conditions is fulfilled:

1. $t_1.colours \cap t_2.colours \neq \emptyset$;

²See Def. 7.2.1 below for a characterization.

2. t_1 and t_2 are opening tags, and $\exists \#C \in t_2.context.radicals$ so that the opening tag $[\#C \text{ over } t_1.name] \in X$;
3. t_1 and t_2 are closing tags, and $\exists \#C \in t_1.context.radicals$ so that the closing tag $\langle \#C \text{ over } t_2.name \rangle \in X$;
4. t_1 is an opening tag and t_2 is a closing tag, and $\exists (\#C, r) \in t_1.context.radicals \times t_2.context.radicals$ so that the opening tag $[\#C \text{ over } r] \in X$;
5. t_1 is a closing tag and t_2 is an opening tag, and $\exists (\#C, r) \in t_1.context.radicals \times t_2.context.radicals$ so that the closing tag $\langle \#C \text{ over } r \rangle \in X$.

Let us now remind that an opening tag of a LeAG relates to the root of the corresponding element in the eAG and, if the name of the tag is associated with a label from \mathcal{L}_{In} of the schema, to the *first* node of that element. Symmetrically, a closing tag to the leaf of the corresponding element in the eAG and, if the name of the tag is associated with a label from \mathcal{L}_{Out} of the schema, to the summit of the *Out* edge of that element. Hence, this corollary suggests a parsing method for ESE defining tags, that is agnostic in terms of the order in which such tags are read by the parser:

ESE defining tags: parsing strategy. Be a schema S , Le a LeAG and X a ToT of Le , whose reference value is r . Be E the corresponding eAG. Let us consider we can create nodes and associate two nodes and a label as an edge of E . Either the root or the leaf of an edge may be undefined.

Considering the tags of X from the ones with the most numerous colours to the ones with the less:

Associate (either by creating it or because it already exists) a node v_o with r as a reference value and typed T_o to each ESE opening tag $Ot \in X$ characterized by the triple $L_oT_oT'_o$ from S , so that there are no two nodes typed T_o with r as a reference value in the hierarchy the element defined by Ot belongs to³. If a rootless edge, characterized by the triple $L_oT_oT'_o$ and whose leaf has r as a reference value and results from a tag belonging to the same hierarchy as Ot , exists, then use v_o as a root for it. Else, create an edge characterized by $L_oT_oT'_o$ and set v_o as its root; if there is an existing node with r as a reference value, whose type is T'_o and that results from a tag from the same hierarchy as Ot , use that node as a leaf. Then, complement any existing, leafless edge characterized by a triple LTT_o , $(L, T) \in \mathcal{L} \times \mathcal{T}$, $L \neq \epsilon$, and whose root results from a tag belonging to the same hierarchy as Ot .

Symmetrically, associate (either by creating it or because it already exists) a node v_c with r as a reference value and typed T'_c to each ESE closing tag $Ct \in X$ characterized by the triple $L_cT_cT'_c$ from S , so that there are no two nodes typed T'_c with r as a reference value in the hierarchy the element defined by Ct belongs to. If a leafless edge, characterized by the triple $L_cT_cT'_c$ and whose root results from a tag belonging to the same hierarchy as Ct , exists, then use v_c as a leaf for it. Else, create an edge characterized by $L_cT_cT'_c$ and set v_c as its leaf; if there is an existing node whose type is T_c , with r as a reference value and that results from a tag from the same hierarchy

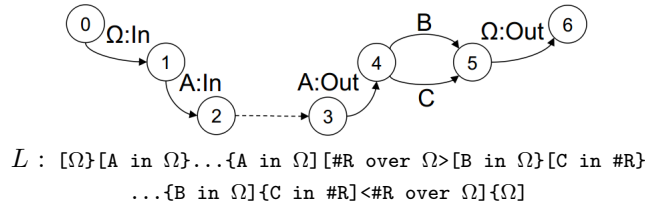
³See Definition 7.2.1 above for a characterization of that fact.

as Ct , use that node as a root. Then, complement any existing, rootless edge characterized by a triple LT'_cT' , $(L, T') \in \mathcal{L} \times \mathcal{T}$, $L \neq \epsilon$, and whose leaf has r as a reference value and results from a tag belonging to the same hierarchy as Ct .

When all the tags have been read, complement the incomplete edges whose label is suffixed either $:In$ or $:Out$ with a new node.

An description of that algorithm along with the appropriate data structures is described in Section 7.3.

Example. Consider the following schema S and LeAG L :



Let us consider the second ToT only here. There are three ESE defining tags in that ToT, plus an opening colour tag :

$$t_1 = \{A \text{ in } \Omega\}$$

$$t_2 = [B \text{ in } \Omega]$$

$$t_3 = [C \text{ in } \#R]$$

$$c = [\#R \text{ over } \Omega]$$

Based on Definition 7.2.1 page 151, on the one hand, t_1 and t_2 belong to a common hierarchy, for they share colours; on the other hand, t_1 and t_3 are on another common hierarchy (see criterion 4 of that definition).

All three ESE tags relate to one colour only, so they are parsed in the order they appear in the LeAG.

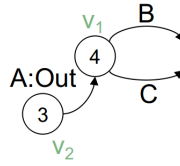
First, t_1 is a closing tag. The name of the tag is A, which relates to the edge characterized by the triple $(A:Out, 3, 4)$. Since no node of type 4 has been defined with the reference value corresponding to the second ToT, a node must necessarily be created. Be v_1 that node. The type of v_1 is 4. There is no leafless edge for the triple $(A:Out, 3, 4)$, so a new edge e_1 , with v_1 as its leaf, is created. No available node can complement the edge.

Next, t_2 is an opening tag. The name of the tag is B, which relates to the edge characterized by the triple $(B, 4, 5)$. An existing node can be associated to t_2 , namely v_1 . An edge e_2 characterized by $(B, 4, 5)$ has to be created with v_1 as its root. No available node can complement the edge.

Eventually, t_3 is an opening tag. The name of the tag is C, which relates to the edge characterized by the triple $(C, 4, 5)$. An existing node can be associated to t_3 , namely v_1 . An edge e_3 characterized by $(C, 4, 5)$ has to be created with v_1 as its root. No available node can complement the edge.

The incomplete edge e_1 , suffixed $:Out$, is completed by a new node v_2 typed 3.

Thus the parsing of the second ToT results in the following subgraph:



The parsing of the ToT illustrated in the above example yields a connected graph. This is only a particular case, resulting from the fact no insert was defined and no ϵ edge was involved within the edges corresponding to the ToT. In general, ϵ edges may be involved within the ToT, as will be detailed in Paragraph 7.2.2. The presentation on how to handle insert tags is given in Paragraph 7.2.3.

A remark must be made here. The above ESE parsing strategy holds only because, in a given ToT, and along a given hierarchy of elements, type values identify nodes, due to the restrictions made onto ESE defining tags.

The same property cannot be extended to insert tags. For instance, consider a comment insert that is declared as a repeatable element in the schema. Then, along a given hierarchy of annotation, that insert may be instantiated twice at the same position, i.e. in the same ToT: then, the type of the root of that insert element is shared by at least two nodes related to the same ToT and involved in the same hierarchy of elements.

In order to maintain the validity of the above strategy for ESE tags, the idea is then to make sensible restrictions on insert tags in order to be able to parse them independently from ESE defining tags. More precisely the goal is to make sure that, in the eAG corresponding to the LeAG, the subgraphs corresponding to the inserts can be (mentally) hidden so that, after their extraction, the connected subgraphs left be the same as those obtained by parsing the ESE tags alone. In such a setting, the ESE tags can be parsed first, and then parsing the inserts yields some other connected subgraphs that, in the final eAG, can be positioned ‘between’ the ESE subgraphs. The conditions for that parsing method are given in Paragraph 7.2.3.

Eventually, in Paragraph 7.2.4, we will discuss how to connect the subgraph obtained by parsing the ToT to the rest of the eAG discovered so far.

7.2.2 Restrictions on ϵ Edges

The first way to connect two disconnected subgraphs yielded by the parsing of some tags from the same ToT is ϵ edges or, more generally, sequences of ϵ edges. The general idea for connecting together the disconnected subgraphs is the following: there is only one leaf in the whole eAG, and hierarchical annotations result in a linear subgraph in the eAG.

Connecting the subgraphs yielded by parsing a ToT relies on the notion of hierarchical level. The following definition clarifies this notion in our context.

Definition 7.2.2: hierarchical level. Be a schema S and a LeAG L . Be x a tag. Be L_x the label associated with x .

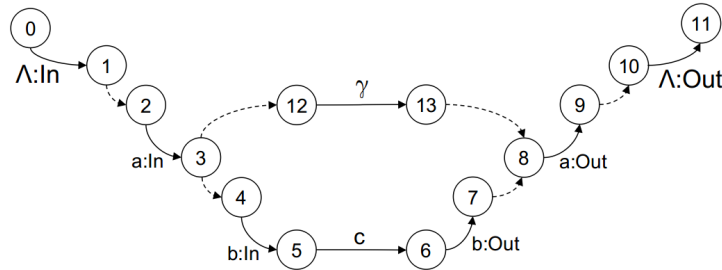
1. if $x = \{\mathbf{n \ in \ } \sum_i r_i\}$, then $\forall i$ the leaf of the corresponding eAG element is in the hierarchical level “in r_i , Left, L”⁴.
2. if $x = \{\mathbf{n \ in \ } \#C\}$ while there is a tag $\langle \#C \ \mathbf{over \ } r \rangle$ in the ToT, then the leaf of the corresponding eAG element is in the hierarchical level “in r , Left, L”.
3. if $x = \{\mathbf{n \ in \ } c\}$, so that $n \in \mathcal{L}_{In}$, then the first node in the corresponding eAG element is in the hierarchical level “in $n\#c.colours$, Right, 1”.
4. if $x = \{\mathbf{n \ in \ } \sum_i r_i\}$, then $\forall i$ the root of the corresponding eAG element is in the hierarchical level “in r_i , Right, R”.
5. if $x = \{\mathbf{n \ in \ } \#C\}$ while there is a tag $[\#C \ \mathbf{over \ } r \rangle$ in the ToT, then the root of the corresponding eAG element is in the hierarchical level “in r , Right, R”.
6. if $x = \{\mathbf{n \ in \ } c\}$, so that $n \in \mathcal{L}_{Out}$, then the last node in the corresponding eAG element is in the hierarchical level “in $n\#c.colours$, Left, Z”.

Nota. The root of the element B corresponding to the ToT $[\#C \ \mathbf{over \ } X \rangle [B \ \mathbf{in \ } \#C\}$ belongs to two hierarchical levels: “in X , Right, R” and “in $\#C$, Right, R”.

ϵ **connections inside a ToT.** Eventually, the nodes belonging to compatible hierarchical levels shall be connected as follows:

may connect to \rightarrow	“in X, Left, L”	“in X, Left, Z”	“in X, Right, R”	“in X, Right, 1”
“in X, Left, L”	-	In priority	2^{n^d} choice	-
“in X, Left, Z”	-	-	-	-
“in X, Right, R”	-	-	-	-
“in X, Right, 1”	-	-	Yes	-

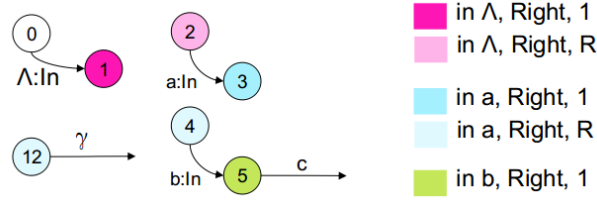
Example. Consider the following ToT and a possible, corresponding schema:



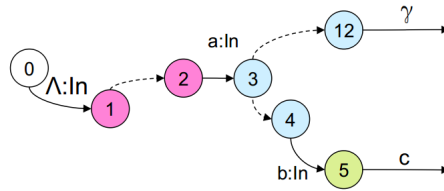
ToT: $[\Lambda]\{a \ \mathbf{in \ } \Lambda\}\{\#R \ \mathbf{over \ } a \rangle [\gamma \ \mathbf{in \ } \#R]\{b \ \mathbf{in \ } a\}[c \ \mathbf{in \ } b] \dots$

Based on the previous paragraph, parsing this ToT will yield the following subgraphs, in which the hierarchical level to which the nodes belong are highlighted:

⁴Here, “Left” means that the hierarchical level in question has opened previously to the current ToT; this serves as a means to disambiguate cases in which an element named X ends in a ToT where another element named X, sharing the same context, starts: each define a hierarchical level “in X”, but one will be “in X, Left” and the other “in X, Right” – see below. ‘L’ stands for ‘leaf’, since it is the leaf of the element related to the tag in question that belongs to the hierarchical level “in r , Left”. Similarly, in the following, ‘R’ will stand for ‘root’, ‘1’ for ‘first’ and ‘Z’ for ‘last’.



Since all hierarchical structures, in an eAG, is represented by a linear graph, it seems quite obvious that we have to gather the subgraphs by connecting, by means of the (sequences of) ϵ edges allowed by the schema, the leaves belonging to a certain hierarchical level to the roots belonging to the same hierarchical level, as follows:



Yet, the above principle needs the following restriction in order to work fine.

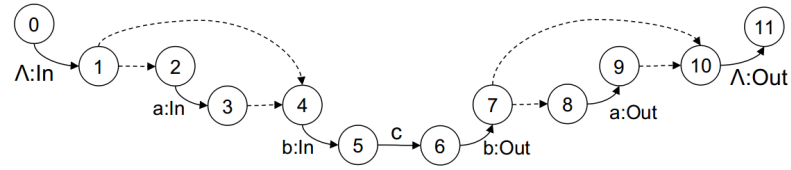
(CE1) The schemas we consider for parsing are ϵ -unambiguous, i.e. if there are two paths \wp_1 and \wp_2 from the root to the leaf of the SeAG that, epsilon edges apart, spell the same, then there is a path \wp_3 that also spells the same, whose set of non-epsilon edges equals both the sets of edges of \wp_1 and of \wp_2 and whose set of epsilon edges is included in the intersection of the sets of edges of \wp_1 and of \wp_2 .

(CE1) is indeed a crucial restriction, since it guarantees that, given a node v_1 of the schema, connected to a node v_2 by a sequence of ϵ edges, then there is one and only one shortest sequence of ϵ from v_1 to v_2 . This also means that it will be possible to deterministically choose one sequence of ϵ edges when connecting two nodes from the eAG during the parsing, namely, that sequence of minimal length.

Note that the very simple connecting strategy described above, which consists in 1) keeping track of the hierarchical levels each node belongs to and 2) connecting the leaves of the subgraphs to the roots that belong to at least one same hierarchical level, is order agnostic:

Property 7.2.2 Be a ToT X from a LeAG Le . Be $G = \{G_i\}_i$ the set of the connected subgraphs of the eAG corresponding to Le . Then the order in which to consider the elements of G for the ϵ connection is indifferent.

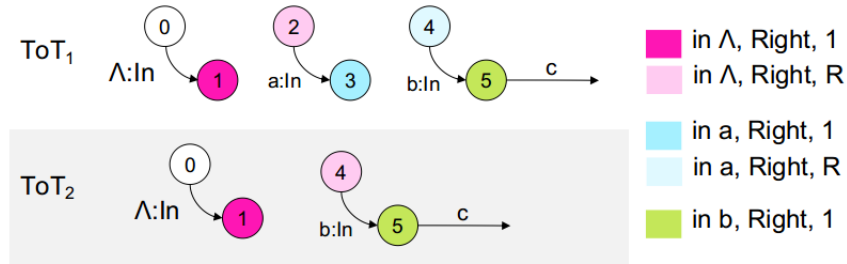
Noteworthy, this algorithm based on hierarchical levels is compliant with hierarchical levels declared as optional in the schema. Let us consider the following SeAG and two possible LeAGs (only their first ToTs are represented here, for the sake of brevity).



$$\text{ToT}_1: [\Lambda]\{a \text{ in } \Lambda\}\{b \text{ in } a\}\{c \text{ in } b\}\dots$$

$$\text{ToT}_2: [\Lambda]\{b \text{ in } \Lambda\}\{c \text{ in } b\}\dots$$

The subgraphs corresponding to the ToTs are the following:



The schema exhibits an optional hierarchical level, which means that the end of the edge $\Lambda:\text{In}$, in an instance of that schema, might be connected to the start of either $a:\text{In}$ or $b:\text{In}$. This example shows that basing the ϵ connections on hierarchical levels is the right solution: in the absence of an opening tag for a , in ToT_2 , the nodes typed 1 and 4 belong to the same hierarchical level, and thus will be connected by the edge $(\epsilon, 1, 4)$ from the schema. On the contrary, when the element a is present (ToT_1), then the nodes typed 1 and 4 are not on the same hierarchical level. Hence, the ϵ edge from 1 to 4 will not be instantiated in the corresponding eAG.

The elements presented so far enable to define an order agnostic, single pass parser for LeAG/SeAG couples that do not contain any insert. In the following paragraph, we study how to consider inserts.

7.2.3 Restrictions on Insert Tags

In the eAG model inserts behave as any element: their label is just marked with the suffix $:\text{Att}$, $:\text{Com}$ or $:\text{LinkTo}$. Yet in LeAG, because of the shift from a graphical to an inline markup model, inserts have some characteristics that set them apart, the most obvious being that they open and close at the same position in the data they are inserted in. This contrasts strongly with ESE elements, defined by two tags separated by at least one character from the primary resources. Because of their span, two ESE elements belonging to the same hierarchy of elements are easy to order: the first one is the one whose ending element is found first along the data. This kind of reasoning does not apply for inserts.

The fact inserts do not have exactly the same status in eAG and LeAG is crucial in designing a parser from LeAG to eAG, obviously. In particular, as will be extensively

discussed below, the fact two inserts sharing the same context, in a LeAG, will have to be ordered and positioned along a hierarchical annotation path in the eAG corresponding to the LeAG, demands new strategies for ordering elements, compared to what was done with ESEs.

Moreover, not all inserts have the same semantics, in order to meet different targets. In particular, there is little similarity between a void insert, whose function is to state the ID of the first node of an element, and a comment insert, that enables to express a structured annotation that is inserted along the annotation. As a consequence, not all inserts will be parsed the same way.

In order to meet our algorithmic goal, several restrictions on the nature of inserts will be defined and commented here.

SeAG/LeAG couples must also verify the following conditions:

(CIcore) In a SeAG, the root of an insert must have an out degree equal to one, and the leaf of an insert must have an in degree equal to one as well.

(CI1) In a SeAG schema, if there is a path connecting two comment inserts⁵, then this path must contain at least one ESE defining edge.

(CI1bis) If there is a path made out of ϵ edges only, connecting the leaf of an insert to its root, then the path must be made out of ϵ edges whose root and leaf are either the root or the leaf of the insert.

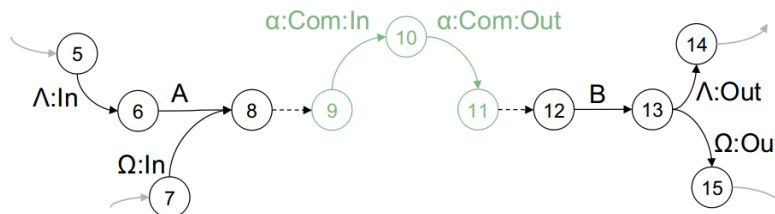
(CI1ter) The only inserts that can belong to an Hamiltonian path containing no ESE defining edge are comment inserts.

(CI2) In a LeAG, the context field of any insert is a single radical (i.e. is either a coloured element name or a colour).

(CI3) In a SeAG schema, an element may be the father of at most one, non-repeatable attribute insert, whose root is the end of the opening `:In` edge of the element.

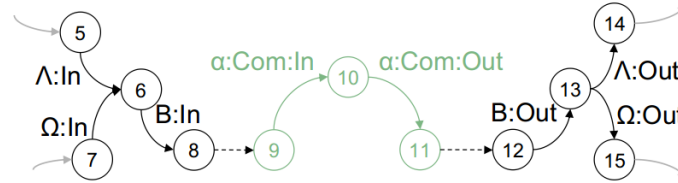
(CI4) In a LeAG, an ESE element contains at most one void insert, which occurs in the same ToT as the opening tag of that ESE.

(CI2) implies that two inserts belong to the same hierarchical level iff their context field is *identical*. Arguably, (CI2) is a strong limitation. Indeed, (CI2) prevents an insert to be included in more than one element or colour. For instance, the following pattern, that is fine in the eAG model, cannot translate into LeAG abiding by condition (CI2), since the insert α is directly included in both Λ and Ω :



⁵Quoting inserts included.

On the contrary, yet, it is worth noting that inserts can very well have several colours, on condition they are included in one and only one element that has the same colours. See for instance, the following eAG pattern may translate into the (incomplete) LeAG L underneath, that respects (CI2):



$L: [\Lambda][\Omega\#C][B \text{ in } \Lambda, \Omega\#C] \dots [\alpha \text{ in } B\#C] \dots \{B \text{ in } \Lambda, \Omega\#C\} \{\Lambda\} \{\Omega\#C\}$

Therefore, the only way for an insert to belong to several hierarchies of elements is to be included in an element that also belongs to the same hierarchies.

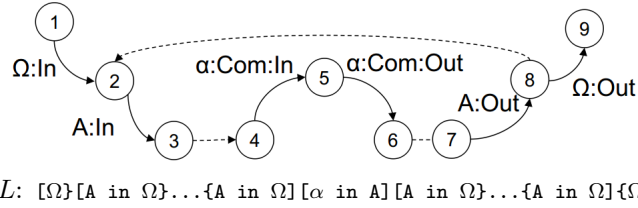
It is unclear whether this restriction, in editorial terms, will be too limiting or not in practise, but it seems that it shall not, considered what inserts represent. Indeed:

1. attribute inserts naturally characterize one element, so restriction (CI2) is quite natural for such inserts;
2. void inserts serve as a means to explicitly state the identifier of the first node of a given element, so the same remark can be made here;
3. link inserts are meant to connect two distinct elements: the one in which the insert is included and another one, whose first node is pointed to by the link. So links will naturally be included in an element.
4. the interest of positioning comment inserts in two disjoint elements is more questionable, from an editorial point of view. Actually, this situation may remind of the fact that a colour cannot be defined over this kind of context either (but can, without restrictions, be defined over a multicoloured element). Only practical experience will help us assess whether those limitations are strong or not.

From an algorithmic point of view, preventing inserts to belong to several hierarchies at the same time has the following consequences. First, it provides a clear and simple characterization of the fact two inserts belong to the same hierarchy of elements (see Property 7.2.3 below), which enables to introduce a syntactical convention for inserts, that disambiguates cases where it is not clear to which element an insert may belong. Second, and more importantly, it provides interesting structural properties (see Property 7.2.4 page 162) that, together with the properties implies by (CI1), enable to parse inserts independently from ESE tags (see Property 7.2.3.3).

Property 7.2.3 Be L a LeAG respecting (CI2), X a ToT from L . Then, two inserts belong to the same hierarchy of elements iff their context part is equal.

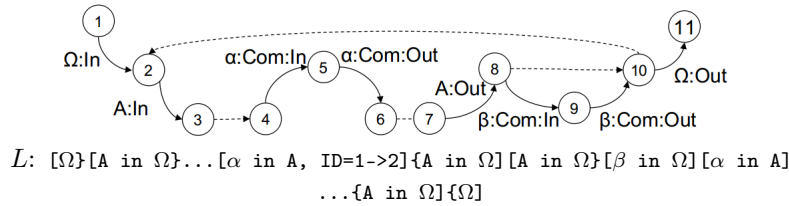
This property enables to introduce a writing convention that enables the editor to handle situations in which it is not clear to which element an insert belongs. The following SeAG-LeAG couple exemplifies those situations:



Indeed, in the LeAG above, it is not possible to assess which element A the insert α is included in. We propose a syntactical rule to enable the editor to assess explicitly if an insert belongs to a closing ESE, an opening ESE, or takes place between a closing and an opening ESE.

Corollary: a convention for writing inserts Be L a LeAG, X a ToT of L . The inserts written on the left of the last non-insert tag in X do not belong to an ESE that starts in the ToT; they are expected to be included in an ESE that ends in X . The inserts written on the right of the last non-insert tag in X do not belong to ESE that ends in the ToT; they are expected not to belong to an ESE that starts in X .

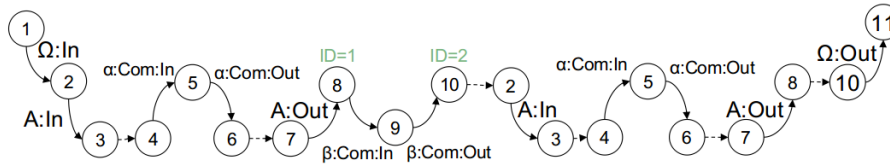
Example. Take the following SeAG-LeAG couple:



In this example, the insert α with the ID field is not included in the element A that starts at the same ToT, while the insert α without an ID field does not belong to the element A that ends in that ToT. Those considerations enable to associate each α insert to the element A they belong to.

The insert β , because of its position in the ToT, is not expected to belong to an element that ends in the ToT. Indeed, it belongs to Ω directly.

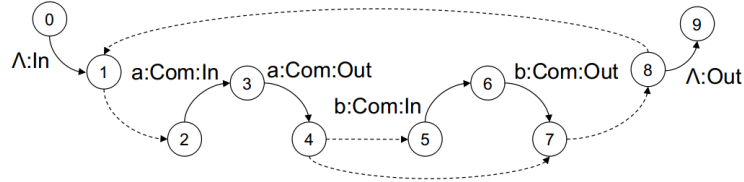
In the end, we can deduce that the above LeAG L shall be associated with the following eAG:



Nota. It is quite clear that the writing convention above only works because an insert cannot be included both in an opening and a closing element. Fortunately, (CI2) prevents from such a situation.

(CI1) means that the only possible sequence made out of comment inserts belonging to the same hierarchical level is a repetition of *the same insert* several times. This restriction, strong as it may seem, is actually both very helpful in keeping the parsing algorithm simple and efficient, and quite logical with the very nature of comment inserts.

Consider the following schema, that does not verify (CI1):



This schema allows an element Λ to contain a sequence of any number of inserts a and b , so that a inserts are at least as numerous as b inserts, and so that two b inserts are separated by an a insert.

To understand how tricky parsing a LeAG against that schema may be, consider the following LeAG:

$$\begin{array}{cccc}
 [\Lambda] \dots & [a \text{ in } \Lambda] \dots & [a \text{ in } \Lambda][a \text{ in } \Lambda][b \text{ in } \Lambda][b \text{ in } \Lambda] \dots & [b \text{ in } \Lambda]\{\Lambda\} \\
 \text{first ToT} & \text{second ToT} & \text{third ToT} & \text{fourth ToT}
 \end{array}$$

Parsing the first ToT is not a problem. Parsing the second shall not be either. Yet parsing the third ToT is way more tricky: we know that the inserts have to be ordered along a path in the eAG, but in what order? By comparing the schema with the list of inserts from the third ToT, it seems that there are two ordering possibilities, namely a, b, a, b or b, a, b, a – yet which is the right one? The question is not easy since the schema allows the insert a defined by the second ToT to be followed either by an insert a or by an insert b , so even considering the previous ToT does not help sort out the right insert sequence for the third ToT... To solve the problem, it is necessary to crawl S in order to determine the set $\mathcal{L}_\Lambda^{ins}$ of all the possible sequences $\mathcal{L}_S^{a,b}$ of inserts a and b allowed by the schema, and then to order the set of all the inserts included in the instance of the element Λ across the different ToTs, so as to find a sequence matching with $\mathcal{L}_S^{a,b}$. To achieve that, once it has been identified that only a and b inserts matter (which demands to list all the inserts occurring in the instance of Λ), the maximal sub-schema whose edges are either ϵ edges or a and b inserts must be identified in S – it is, in this particular case, the subgraph defining the interior of Λ . That subschema must then be interpreted as an automaton defining a language whose letters are inserts – in this particular case, the language is $\mathcal{L}_S^{a,b} = (a \cdot b?)^*$. Then, the inserts of each ToT containing several inserts belonging to the same hierarchy, in the LeAG, must be ordered into a sub-word so that the concatenation of the subwords, in the order of the corresponding ToTs, be matched against a word of the language of the automaton that was discovered... In the above example, it is necessary to take into account the list of inserts from the *fourth* ToT to sort the right insert sequence

for the third ToT: since the second ToT yields a , the third a, b, a, b or b, a, b, a and the last b , then only the following concatenation results in a word from $\mathcal{L}_S^{a,b} = (a \cdot b)^*$, that is a, b, a, b, a, b ⁶. On the contrary, condition (CI1) will enable to stick to a very simple parsing algorithm for inserts, based on a single pass (see below).

Moreover, it is not contradictory with the nature of inserts not to allow the definition, in the schema, of intricate patterns made out of insert elements. A pattern from a schema enforces some elements to appear in a given order; yet since inserts take no room along the primary resources, what does the relative order of two insert elements occurring at the same reference point mean?

Additionally, from a practical point of view, it would not be easy for the end-user (i.e. the annotator) to maintain her annotation consistent with the schema, would it not respect (CI1): as the sketched algorithm above shows, the set of insert tags allowed in a given ToT depends on the previous ToT, and limits the choice of inserts for future positions, in a way that may not be easy to grasp⁷. This seems to indicate that restriction (CI1) is a sensible one, apart from algorithmic considerations.

Nota Bene. Thus, since the only sequences of comment inserts that we may face will be the same insert from the schema instantiated multiple times at the same position, we need a criterion to order the insert instances. The order of appearance of the insert tags along the LeAG will be just fine.

To finish with, (CI1bis) simply means that an insert can be both optional and repeatable, but the way to express each characteristics is unique: optionality, by connecting the root of the insert to its leaf by means of one ϵ edge; multiplicity, by connecting the leaf of the insert to its root by means of one ϵ edge too.

(CI1ter) reinforces (CI3) and (CI4), that prevent attribute inserts and void inserts to occur multiple times in the same element in a LeAG: (CI1ter) bans cycles, in the SeAG, that shall contain only inserts but not only comment inserts.

The above restrictions (CI1), (CI1bis), (CI1ter) and (CI2) imply the following property.

Property 7.2.4 Be a LeAG L , a ToT X of L . At this step, we consider there is no link or quote insert in L . Be I_C the set of inserts that occur, in X , on the left of the

⁶Moreover, it is quite easy to imagine LeAGs for which not only one, but several orderings of the inserts match the schema, without there being any reasonable way to pick one over the others. Take, for instance, the following annotation: $\Lambda\}\dots [a \text{ in } \Lambda]\dots [a \text{ in } \Lambda][a \text{ in } \Lambda][a \text{ in } \Lambda][b \text{ in } \Lambda]\dots [b \text{ in } \Lambda]\{\Lambda$. It is possible to make the following words out of the inserts' names as sketched above: a, b, a, a, a, b ; a, a, b, a, a, b and a, a, a, b, a, b . Indeed, one may argue that since all those sequences are acceptable, one could pick any indifferently. Yet in order for the parser to behave deterministically, we need to establish arbitrary rules that ensure the chosen sequence will be the same any time the parser runs on the same data.

⁷E.g. with the above schema, after a ToT containing more bs than as , an a must necessarily occur...

first non-insert tag, if such a tag exists, and if so, be I_O the set of inserts that occur, in X , on the right of the last non-insert tag. We have the following results:

1. Each comment insert of L can be associated with a rooted, single-leafed graph.
2. The set of comment inserts of I_C that share the same context r constitutes an ordered set $[a_i^r]_i$, so that either $root(a_{i+1}^r) = leaf(a_i^r)$ or there is a path made out of ϵ edges only connecting $leaf(a_i^r)$ to $root(a_{i+1}^r)$. Let us denote \wp_C^r the subgraph made out of the comment inserts from I_C of context r and the ϵ edges. Idem for the comment inserts from I_O . Let us denote \wp_O^r the subgraph made out of the comment inserts from I_O of context r and the ϵ edges.
3. Be $\wp_C^{r_1}$ and $\wp_C^{r_2}$, $r_1 \neq r_2$. Then L can be parsed so that $\wp_C^{r_1}$ and $\wp_C^{r_2}$ have either no node in common or, at most, their leaf.
Be $\wp_O^{r_1}$ and $\wp_O^{r_2}$, $r_1 \neq r_2$. Then L can be parsed so that $\wp_O^{r_1}$ and $\wp_O^{r_2}$ have either no node in common or, at most, their root.
4. More generally, L can be parsed so that all the roots and leaves of the comment inserts have a degree equal to 2, apart from the nodes that also serve as the root or leaf for the \wp graph the insert they constitute belongs to: the degree of those nodes may be more than 2.

Proof. 1. This is true of any element in an eAG.

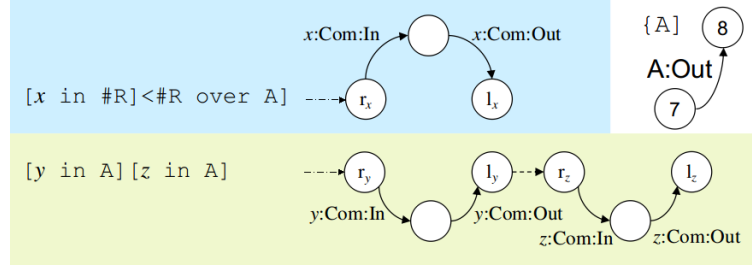
2. Two inserts sharing the same context belong to the same hierarchy of elements and, more importantly here, to the same hierarchical level of that hierarchy. Because no element of that hierarchy but an insert may start and end at the same reference point, then two inserts of context r can be separated but by one or more inserts (plus optional ϵ edges) whose context contains r . Since inserts' contexts are limited to a single radical, then the inserts whose context contains r are reduced to those whose context *is* r .

3. The question here is: can a LeAG be parsed in a way that prevents inserts to belong to more than one chain of inserts \wp_C^r . The first case in which an insert should have belonged to two chains of inserts is exemplified by the following LeAG:

$$\dots [a \text{ in } X] [a \text{ in } Y\#R] [a \text{ in } X, Y\#R] \dots$$

Yet, this situation is forbidden by (CI2) which states that the context field of any insert is a single radical. Thus, the only case in which two chains $\wp_C^{r_1}$ $\wp_C^{r_2}$ may converge is when there is a colour name $\#R = r_1$ so that there is a closing colour tag $\langle \#R \text{ over } r_2 \rangle$ in the ToT. For instance, the following figure shows the graphs that should result from the parsing of a LeAG containing three inserts, two belonging to A directly, and one belonging to a colour $\#R$ over A (x, y, z denote label variables, $r_x, l_x, r_y, l_y, r_z, l_z$ type variables):

$$\text{ToT: } [x \text{ in } \#R] \langle \#R \text{ over } A \rangle [y \text{ in } A] [z \text{ in } A] \{A\}$$



In this case, because it is the last element of a colour built over A that comes to an end, it is quite clear that the leaf of the insert x will have to be connected to a node of the hierarchical level of the elements directly included in A . The question is: are there situations in which this node cannot be neither the leaf of the insert z nor the summit of $A:Out$?

First, (CI1) implies that $y = z \wedge r_y = r_z \wedge l_y = l_z$. If the LeAG is valid, we know that there is a way to connect the node of type l_z to the summit of $A:Out$, either because $l_z = 7$ or by means of one or more consecutive ϵ edges.

Now, if $x = y = z$, then $l_x = l_z$, so there surely is a way to connect the leaf of x to $A:Out$ directly. In that case, the convergence of $\#R$ and the hierarchical level under A occurs at the summit of $A:Out$.

Eventually, let us consider now the case $x \neq y$. Then because of (CI1), $l_x \neq r_z$ and there is no sequence of ϵ edges that connect l_x to r_z in the schema either, so the leaf of x cannot connect to the chain of inserts $y - z$ on the nodes typed $r_z = r_y$. Thus, the connection will occur either on the nodes of type l_z or on the summit of $A:Out$. In the first case, we can choose to make that connection on the node of type l_z that works as a leaf for the insert z , i.e. the leaf of \wp_C^A .

Idem for the comment inserts in I_O .

4. As a consequence of 3, it is possible to define a parsing strategy so that if two inserts belong to a given path within the subgraph resulting from the parsing of the set of comment inserts, then they have the same context. Be r a context, \wp_C^r the corresponding chain of inserts. In \wp , the only nodes that belong to the hierarchical level immediately under r have a type value equal either to the root or to the leaf of the inserts \wp_C^r , thanks to (CI1bis).

Parsing the inserts of I_C only yields a connected subgraph of the final eAG. As stated before, all the connected subgraphs yielded when parsing the different sets of tags of a ToT will have to be brought together into one connected graph that will, in turn, be connected to the eAG under construction. The criterion used to pick the node of each subgraph that will serve as a connection point with another subgraph will be its type, its colour and/or the hierarchical level it belongs to (see Paragraph 7.2.2). Since, under those criterion, any node of \wp_C^r is undistinguishable either from the root or of the leaf of \wp_C^r , then it means it will always be possible to make the connections with the other subgraphs, i.e. with the other elements of the eAG, at the root or leaf of the chains of inserts. \square .

To finish with, let us comment on the restriction (CIcore). It is the core restriction, in that it guarantees the following property, described earlier page 154, that enables

to parse ESE defining tags and insert tags separately.

Property 7.2.5 Be S a schema, L a LeAG. S respects (C_{icore}) implies that the connected subgraphs obtained by parsing the ESE tags of each ToT of L alone are included in the eAG corresponding to L .

Proof. It is quite clear that in a LeAG containing only ESE, all the subgraphs corresponding to the parsing of the ESE tags of the ToTs are included in the eAG. The discrepancy may then occur because of the presence of inserts.

Be X a ToT containing inserts, $\{G_i^{ESE}\}_i$ the set of subgraphs resulting from the parsing of the ESE tags only, and G^{real} the connected subgraph resulting from the parsing of X .

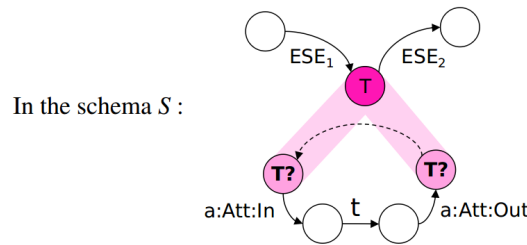
The presence of each ESE tag, in a ToT, implies the presence of one and only one edge, whose label matches the name of the ESE tag (see restriction (C1) for a definition of this matching), both in G^{real} and in a given G_k^{ESE} – the contrary would contradict the general parsing principles given in Paragraph 7.1. Thus we can establish that there is a bijection ϕ from the set of edges of $\{G_i^{ESE}\}_i$ to the set of ESE defining edges of G^{real} . It follows that $\exists k \mid G_k^{ESE} \not\subset G^{real}$ implies that there are two edges e'_1, e'_2 of G_k^{ESE} so that $leaf(e'_1) = root(e'_2)$ and $leaf(\phi(e'_1)) \neq root(\phi(e'_2))$.

Since the cause of $G_k^{ESE} \not\subset G^{real}$ is the presence of inserts, it follows that there is a subgraph I_{12} connecting $leaf(\phi(e'_1))$ to $root(\phi(e'_2))$ in G^{real} , containing a hierarchical annotation path made out of insert elements only.

Yet $leaf(e'_1) = root(e'_2)$ implies that the two nodes have the same type value T ; condition (C1) implies that $leaf(e'_1)$ and $root(e'_2)$ also have the same type value. This implies that, in the SeAG, the insert elements are included in a Hamiltonian cycle, which clears out the possibility of there being any thing but comment inserts among the insert sequence because of (C_{iter}): the inserts are then several instances of the same comment insert (see (C_{i1})), denoted Ins hereafter.

Additionally, the Hamiltonian cycle is so that there is a node of that cycle, the node of type T that is also the ending node for an ESE defining edge and the start of another ESE defining edge.

Because of restrictions (C1) and (C_{i1bis}), the shape of the cyclic subgraphs containing the definition of a comment insert element is well-known: there is one epsilon edge connecting the leaf of the insert to its root. Thus the node of type T is a node belonging to Ins (not to ϵ edges only, for instance). The schema configuration implied by $G_k^{ESE} \not\subset G^{real}$ can thus be illustrated as follows:

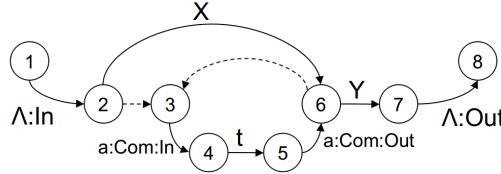


As evidenced by the above illustration, the node of type T cannot be but the root or the leaf of the insert in the schema – any other node resulting in the impossibility of

having a hierarchical path made out of the edge ESE_1 , then the Hamiltonian path starting from the node of type T , and then ESE_2 .

As a conclusion, $G_k^{ESE} \not\subseteq G^{real}$ implies that the in-degree of the leaf of the insert, or the out-degree of the root of the insert, is not equal to one. \square

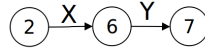
Counter-example. Consider the following SeAG S , that contradicts (CICore)⁸.



Then consider the following LeAG, made out of ESE only divided into three ToTs:

$$L_1 : [\Lambda]\{X \text{ in } \Lambda\} \dots \{X \text{ in } \Lambda\} [Y \text{ in } \Lambda] \dots \{Y \text{ in } \Lambda\} \{\Lambda\}$$

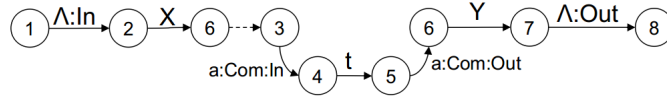
Parsing the second ToT will yield the following connected subgraph G_1^2 :



Now consider the following LeAG, which this time contains the same ESE elements plus an insert, as follows:

$$L_2 : [\Lambda]\{X \text{ in } \Lambda\} \dots \{X \text{ in } \Lambda\} [a \text{ in } \Lambda] [Y \text{ in } \Lambda] \dots \{Y \text{ in } \Lambda\} \{\Lambda\}$$

The only eAG corresponding to the L_2 , given the SeAG S above, is the following graph G_2 :



Since G_2 does not contain G_1^2 , we can conclude that, with this particular schema S , that does not respect (CICore), it is not possible to parse ESE defining tags and inserts independently.

On the contrary, SeAG abiding by (CICore) enable to parse first the ESE tags, and then the insert tags.

Hence the following parsing strategy.

⁸This schema describes an element Λ that ends by an element Y , so that Y may be preceded by a series of inserts, the last insert being positioned where Y starts, and so that the part of λ that is not qualified as Y may be qualified as X ... This kind of description, because it is disqualified by (CICore), is not allowed in the perspective of the parsing. Only a user study would show whether this disqualification is problematic or not, from an editorial point of view. Hopefully, it is baroque enough not to...

7.2.3.1 Comment Insert Tags: Parsing Strategy.

Be S a SeAG L a LeAG respecting the above conditions, so that L contains only ESE and comment inserts tags. Since it is possible to parse the inserts of L so that they form connected subgraphs whose only nodes that may be connected with nodes from other elements than the inserts that belong to the subgraph in question are its root and leaf, the ToT of L can be parsed as follows:

First the ESE tags must be parsed as indicated before. This operation yields a set of connected subgraphs $G = \{G_i\}_i$. The hierarchical levels to which the leaves and nodes of those connected subgraphs belong must be registered (see 7.2.2 page 154).

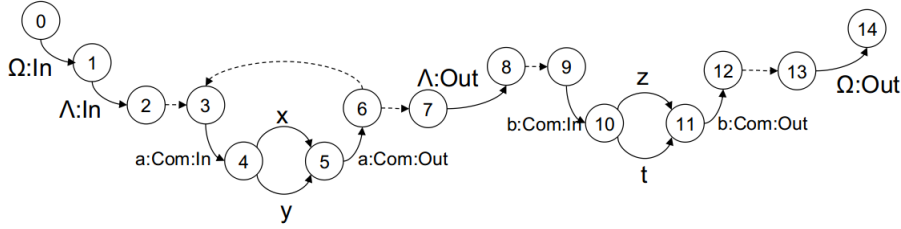
Then, the inserts must be partitioned: first, into two groups I_C and I_O , the first group containing those that occur on the left of the last non-insert tag in the ToT, and the second containing the others; second, in each group, by context field value. Each final set will translate into a sequence of comment inserts, two consecutive inserts being connected by the minimal ϵ sequence between their leaf and root as allowed by the schema. To achieve that, first, parse the LeAG field of the first insert as if it were an independent LeAG, then augment it with a pair of edges labelled with the name of the insert and the suffix `:Com:In` and `:Com:Out`; then treat the next insert identically before connecting it to the previous one, etc. This operation yields a set of connected subgraphs $I_C = \{I_C^r\}_r$ and $I_O = \{I_O^r\}_r$ characterized each by the context r they belong to.

Eventually, the root and leaf of each subgraph I_X^r , $X \in \{C, O\}$ have to be connected to nodes belonging to the same hierarchical level as follows:

1. For the elements of I_C :
 - (a) necessarily connect the leaf, and try to connect the root, of I_C^r , to two nodes that belong to the hierarchical level “in r , Left, Z” and “in r , Left, L” respectively. This case relates to insert chains that belong to an element that closes in the ToT.
 - (b) OR try to connect the root of I_C^r to a node of the hierarchical level “in r , Left, L”, and the leaf of I_C^r to a node of “in r , Right, R”. This case relates to insert chains that belong to an element that neither closes nor starts in the ToT.
2. For the elements of I_O :
 - (a) necessarily connect the root, and try to connect the leaf, of I_O^r , to two nodes that belong to the hierarchical level “in r , Right, 1” and “in r , Right, R” respectively. This case relates to insert chains that belong to an element that starts in the ToT.
 - (b) OR same as 1(b) above.

In the end, for a LeAG containing only ESE and comment inserts, this procedure yields the subgraph corresponding to the ToT.

Example. Consider the following SeAG/LeAG pair.



ToT : ... [a in Λ; [x]...[x]] [b in Ω; [t]...[t]] [a in Λ; [y]...[y]] {Λ in Ω}

Parsing the ESE tags yields the following subgraph, reduced to a single edge:



Then, the comment inserts are partitioned as follows:

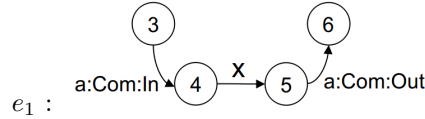
$$I_C = \{ [a \text{ in } \Lambda; [x] \dots [x]] [b \text{ in } \Omega; [t] \dots [t]] [a \text{ in } \Lambda; [y] \dots [y]] \}$$

containing $I_C^\Lambda = \{ [a \text{ in } \Lambda; [x] \dots [x]] [a \text{ in } \Lambda; [y] \dots [y]] \}$

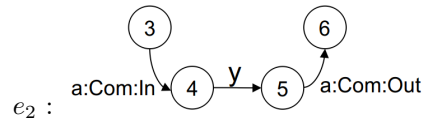
and $I_C^\Omega = \{ [b \text{ in } \Omega; [t] \dots [t]] \}$

$$I_O = \emptyset$$

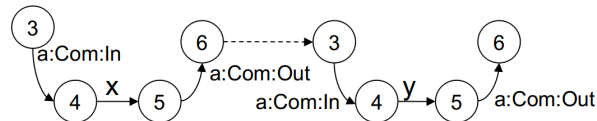
Then, let us consider I_C^Λ . Its first constituent is $[a \text{ in } \Lambda; [x] \dots [x]]$. First, the LeAG field must be parsed, and the eAG resulting from this parsing be augmented with the edges delimiting the insert element, which yields the following graph:



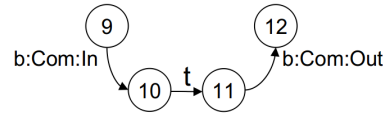
The same must be done with the second insert of I_C^Λ . This yields:



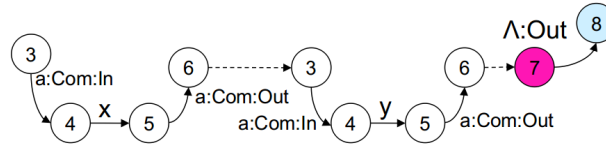
Then the root of e_2 , which is of type 6, must be connected to the leaf of e_1 , which is of type 3. The shortest sequence of ϵ edges connecting a node typed 6 to a node typed 3 is a single ϵ edge (which is OK with (CI1bis)) characterized by the triple $(\epsilon, 6, 3)$. This edge can be instantiated with the leaf of e_1 as a root and the root of e_2 as a leaf. This yields the following graph, denoted I_C^Λ :



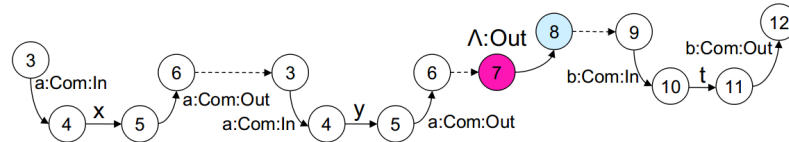
The same procedure then applies to I_C^Ω , which yields the following homonym graph:



Eventually, those graphs want connecting with e_0 . First, we investigate whether the leaf of I_C^Λ can connect the root of e_0 , which belongs to the hierarchical level “in Λ , Left, Z”. The answer is it can, by means of $(\epsilon, 6, 7)$. This ϵ edge is thus instantiated. The root of I_C^Λ does not find a corresponding node, on the contrary. So we are done with I_C^Λ . This builds the following graph:



The only match for I_C^Ω is for its root, which can connect to the leaf of e_0 . This produces the following, connected graph that corresponds to the whole ToT:



To finish with, the following describes how quote, attributes, void and link inserts can be parsed.

7.2.3.2 Quote Insert Tags: Parsing Strategy.

In the LeAG model, quote inserts are a particular case of comment inserts, whose LeAG field is restricted to a pair of node identifiers id_1 and id_2 (it is, more formally, a second ID field). The similarity between comment and quote inserts is reinforced here since both kinds of inserts must abide by the same restrictions – see (CI1ter) and (CI2). Hence, parsing a quote insert is quite easy:

1. Quote inserts can be parsed alongside other comment inserts. In particular, they can be partitioned into the same groups I_C and I_L , and then separated by their context field’s value (see page 167).
2. Then, for each quote insert of each set of quote inserts, associate (or create) two nodes v_1 and v_2 with the identifiers id_1 and id_2 respectively specified in the second ID field of the insert.
3. Create a pair of edges labelled with the name of the insert and the suffix **:Com:In** and **:Com:Out**. The start of the **:Com:In** edge is the root of the insert, its end is v_1 ; the end of the **:Com:Out** edge is the leaf of the insert, its start is v_2 .

4. Proceed with the resulting disconnected graph made out of the two previously defined edges as a comment insert, without worrying about connecting it with the rest of the eAG. This shall happen ‘mechanically’ when the identifiers id_1 and id_2 will be encountered in the ID field of some other tags defining an element, if they have not been found yet.

7.2.3.3 Attribute Insert Tags: Parsing Strategy.

Attribute inserts distinguish themselves from comment inserts, structurally speaking, by not being repeatable, and by having to share their root with the first node of the element they are included in.

This suggests that attribute inserts shall be parsed before comment inserts: Be X an element, a the attribute insert of X . Once the ESE tags of the ToT in which X starts have been parsed, a is parsed just like a comment insert, with the notable exception of having its delimiting edges suffixed `:Att0:In` and `:Att0:Out`; then its root is connected to the first node of X . Then its leaf will serve as a connection node onto which the elements contained in X , inserts or others, will connect.

7.2.3.4 Void Inserts: Parsing Strategy

Void inserts are a syntactic tool for stating explicitly the ID of the first node of an element. It is better to leave void inserts as the last inserts to parse: parsing them simply means checking if there is already a node with the ID they bear and, if so, getting the type of that node; finding the first node of the element the void insert belongs to, and if the type is compatible with the type found previously, give this node the ID specified in the insert. Else, notify a parsing problem.

7.2.3.5 Link Inserts: Parsing Strategy

Link inserts are quite particular since their root belongs to the element they are included in, while their leaf can be a node of any element in the annotation.

Link inserts shall be parsed just before void inserts⁹, that is, after comment inserts. The parsing strategy for link inserts is the following:

1. If it is not empty, parse the LeAG field of the insert. This gives a rooted, single-leafed connected graph E_{in} .
2. Find, in the schema, the triple LTT' associated with the name of the insert. Get the type t_r of its root.
3. Look for the node of type t_r and belonging to the same hierarchical level as the insert, among the nodes related to the current ToT. If it does not exist, notify a parsing problem; else, choose that node as the root of the linking element.
4. Associate a node to the identifier specified as a target in the insert. Choose that node as a leaf for the linking element.

⁹Void inserts, since they do not define any node or edge but only enforce the identifier value of some node defined by another tag, are the last tags to be parsed.

5. If E_{in} is defined, augment E_{in} with a pair of edges labelled with the name of the insert, suffixed $:In$ and $:Out$, so that the root and leaf of the resulting element be the nodes identified previously as such. If E_{in} is undefined, simply create an edge labelled with the name of the insert.

7.2.3.6 Summary

As a summary of Paragraphs 7.2.1, 7.2.2 and 7.2.3, we have seen how to obtain a graph representing a ToT of a LeAG, containing ESE defining tags and insert tags. We have established that, under the appropriate conditions, ESE defining tags and insert tags could be considered separately.

- First, parsing the ESE tags of a given ToT yields a disconnected graph that contains either complete edges or incomplete edges, i.e. edges whose root or leaf has not been defined yet. An incomplete edge whose leaf (resp. root) is missing is said to contain a pending root (resp. leaf). Complete edges will result from tags associated with edges suffixed $:In$ or $:Out$; incomplete edges result from tags that either open or close elements corresponding to unsuffixed edges. Let us denote G^{ESE} this potentially disconnected graph.
- Second, attribute inserts can be parsed, each independently from the others. Since the root of an attribute element must be the first node of the element the attribute qualifies, the tag resulting from the parsing of each insert must connect to the corresponding node of G^{ESE} . Let us denote $G^{ESE+Att}$ this potentially disconnected graph.
- Third, comment inserts can be parsed. They are grouped by the element they belong to. Each insert tag can be parsed, independently from the others, which yields a graph corresponding to the comment element. Then the graphs corresponding to a group of inserts are connected together so that they form a chain (the leaf of an element being connected to the root of at most one another element, and conversely), leaving one root and one leaf of insert elements non connected. When the comment insert is a quote insert, and if, in that case, the quoted element has not been described by tags from the previous ToTs, then the parsing of the quote insert results in a pair of edges, one pointing towards a node that will be the root of the quoted element, one starting from a node that will be the leaf of the quoted element. Those two nodes, structurally speaking, appear as a leaf and a root in the graph resulting from the parsing of the ToT. We call them a target root and a target leaf, to differentiate them from the other roots and leaves of the graphs defined by the ToT. Let us denote $G^{ESE+Att+Com}$ this augmented, potentially disconnected graph.
- Fourth, link inserts can be parsed. The root of the linking element is an existing node of the element the link is defined in; the leaf of the link a node characterized by its identifier. We call that node a target leaf. Let us denote $G^{ESE+Att+Com+Link}$ this still potentially disconnected graph.

- Fifth, void inserts are parsed. Parsing a void insert means specifying the identifier of the first node of the element the insert belongs to. The graph resulting from the parsing of the ToT, after void inserts have been taken into account, is still $G^{ESE+Att+Com+Link}$, with modified node identifiers.
- Last, ϵ edges must be defined in order to connect together the pairs of roots and leaves from $G^{ESE+Att+Com+Link}$ that match, based on the hierarchical levels they belong to. This operation results in a final graph subsuming $G^{ESE+Att+Com+Link}$.

Let us denote G_i the final graph resulting from the complete parsing of the i^{th} ToT of a LeAG. Be z_L the total number of ToTs in a LeAG L .

From now on, the goal is to obtain a single rooted, single leafed connected graph at the end of the whole parsing process. First, this requires to check that each node playing the role of a target node in a given G_i appears in another G_l , $l \neq i$, as a non target-node. This also means to connect each graph G_i , $i > 1$, to the previous $\{G_k; k < j\}$, in order for the hierarchical paths interrupted in a graph G_k and continued in G_i to be completed, and to appear as a path, as it should, in the final eAG. This task of connecting G_i , $i > 1$ to $\{G_k; k < j\}$ is divided into several cases. First, as will be detailed in Paragraph 7.2.5, new and ending colours may introduce connections between ToT graphs: indeed, when a coloured graft begins by an epsilon edge, there may be a difference between the reference value of the beginning of the graft and of the beginning of the first element of the graft. This discrepancy results in the need to connect the ToT corresponding to the beginning of the graft to the one where the first element of that graft appear. The same applies, symmetrically, to ending colours. Second, is the more general case of ongoing hierarchies of elements, that were represented by a path in a given G_k and in G_i , $i > k$, but not in any G_j , $k < j < i$. Two cases there: either the last edge of that hierarchical level, in G_k , was a complete edge, or it was an incomplete. We study those two cases in Paragraph 7.2.4 first, before considering the more intricate situation of opening and closing colours.

7.2.4 Connecting ToT Graphs: Ongoing Hierarchies of Elements

The problem here is to make sure that the last node involved in an element of a given layer of annotation, belonging to G_i , the ToT graph associated with the i^{th} ToT of a LeAG L , be connected to the next node involved in an element belonging to that hierarchy.

In LeAG, an annotation layer is characterized by a colour. Be $\#c$ that colour. We have the following result:

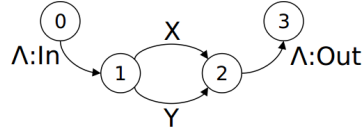
Be a ToT T_i in which colour $\#c$ appears in the context field of at least one element defining tag. Then if there is one ESE opening tag in T_i that is associated to a triple LTT' so that $L \notin \mathcal{L}_{In} \cup \mathcal{L}_{Out}$, then the last node of the hierarchy $\#c$ is a pending root. That node will then connect to the next node of the hierarchy $\#c$, which will have to be a pending leaf, by means of an edge labelled L .

In all other cases, the last node of $\#c$ will not be a pending node and will be connected to the next node of $\#c$ by means of ϵ edges. The detail of each procedure follows.

7.2.4.1 Pending Nodes

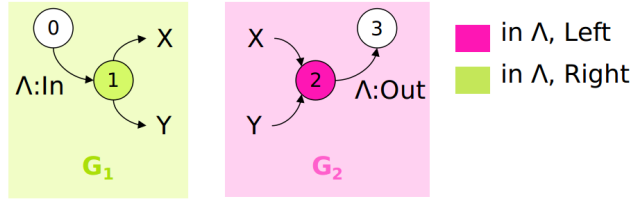
The case of pending nodes is quite simple. Two nodes match iff they are respectively the root and the leaf of an incomplete edge having the same label, and if there is at least one common colour $\#R$ between the names of the hierarchies they belong to.

Example. The following SeAG validates the subsequent, elementary LeAG.



LeAG: $[\Lambda][\#R \text{ over } \Lambda] > [Y \text{ in } \Lambda][X \text{ in } \#R] \dots \{Y \text{ in } \Lambda\}\{X \text{ in } \#R\} < [\Lambda]$

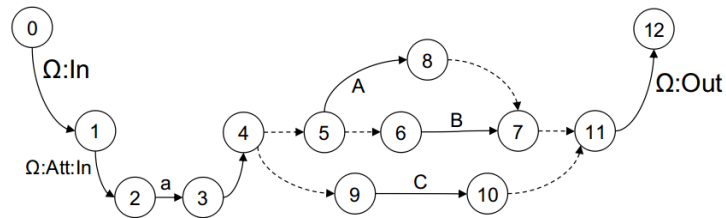
The two ToTs translate into the following graphs:



Obviously, gluing the incomplete edges results in an eAG that is, in this particular case, homomorphic to the schema.

7.2.4.2 Non-Pending Nodes

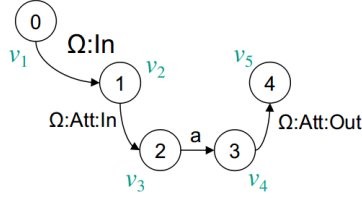
Then, the procedure to connect two nodes of the same colour is more complicated. Let us take an example to illustrate that. Consider the following schema.



Now, consider the following LeAG:

$[\Omega][\text{Att of } \Omega; [a] \dots [a]] \dots [B \text{ in } \Omega] \dots \{B \text{ in } \Omega\}\{\Omega\}$

Parsing T_1 , the first ToT of that LeAG, yields the following graph G_1 :

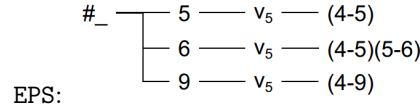


The graph G_1 represents the beginning of a single annotation layer, that is uncoloured. The last node of that layer is v_5 ¹⁰. It is not a pending root. Since T_1 is not the last ToT of the LeAG, this means that v_5 will be the root of a sequence of ϵ edges connecting v_5 to the next node of the uncoloured layer.

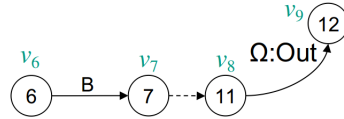
By looking at the schema, there are three different possibilities for that sequence of ϵ , depending on the (unknown, at this step of the parsing) nature of the next uncoloured element in the eAG:

- if the next element is A : the ϵ sequence will be (in terms of LTT' values) $[(\epsilon, 4, 5)]$.
- if the next element is B : the ϵ sequence will be $[(\epsilon, 4, 5)(\epsilon, 5, 6)]$.
- if the next element is C : the ϵ sequence will be $[(\epsilon, 4, 9)]$.

Those information about the possible sequences of ϵ assuring the inter ToT connection for the uncoloured hierarchy will have to be stored in a tree-structure EPS with the following levels: Colour/Type of the attainable node/Source node/Detail of the ϵ sequence. For the above example, that would give:



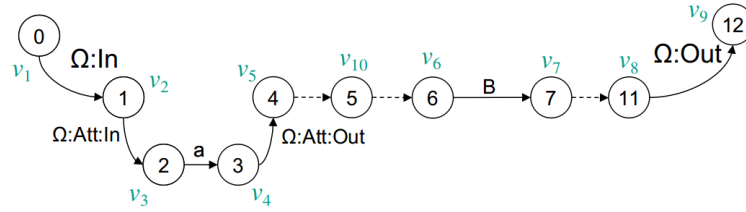
Then, parsing the second ToT yields the following graph:



The first node of the layer is v_6 . Since its type is 6. EPS/#_/6 gives the value: v_5 . Thus the ϵ sequence has to be instantiated between the node v_5 and the current node v_6 . EPS/#_/6/ v_5 gives $[(4-5)(5-6)]$: thus the ϵ edges to be instantiated are ϵ_{45} , rooted on v_6 and ending on a new node v_{10} , followed by ϵ_{56} , rooted on v_{10} and ending on v_6 . The reference value of v_{10} could be any value in the $[ref(v_5); ref(v_6)]$ range; conventionally, we take the reference value of the last node, i.e. $ref(v_6)$.

Since there is no other hierarchical level in this eAG, and since there is no further ToT, this complements the eAG corresponding to the LeAG as follows:

¹⁰The fact the node belongs to that layer can be deduced from the register of the hierarchical levels it belongs to.



Nota. Obviously, all the values of the EPS structure corresponding to a given hierarchical level must be reset after each ToT defining at least one tag from that hierarchical level.

7.2.5 Connecting ToT Graphs: Colour Tags Handling

An annotation layer is, both in eAG, SeAG and LeAG, a hierarchy of elements. An element X of an annotation may contain two layers of annotation (in which case the overall annotation is not hierarchical any more) when more than one hierarchical annotation is defined as the content of the element X .

In eAG, hierarchical annotation takes the shape of a path, respecting certain syntactical constraints (see the presentation of the eAG data model). Thus, multilayering is characterized by the presence, between two given nodes of the element X , of two (or more) parallel hierarchical annotation paths.

We remind here that we differentiate between two kinds of multilayering. Schema-based multilayering is when there are several parallel, disjoint paths inside the schema that can be instantiated each in one of the parallel paths from the instance. All the other situations belong to simulation-based multilayering. In particular, simulation-based multilayering enables to instantiate the same path of the schema several times in parallel, as illustrated by the annotation of anaphoric chains shown previously.

By contrast, a LeAG is composed of an underlying, uncoloured hierarchical layer, onto which grafts plug (grafts that themselves can serve as a basis for further grafts, and so on). A graft is characterized by its colour, a context element into which it is enclosed, and a hierarchy of elements it is composed of. The analogy with eAG is quite clear: two parallel paths in an eAG corresponding to a two-layer LeAG are 1) the path describing the uncoloured content of the element that serves as a context for the graft and 2) the coloured content of the graft itself. The two paths diverge one from the other on a given node that is involved in the context element – i.e. the **starting node of the graft** – and converge on another node, also involved in the context element – i.e. the **ending node of the graft**. For instance, in Figure 7.1.(a) page 176, the starting point of the graft is the node v_4 and its ending node v_9 .

Noteworthy, the starting node of a graft and the **root of the first element** of the graft can differ, and symmetrically for the ending node of the graft and the **leaf of the last element** of the graft. For instance, in Figure 7.1.(a), the root of the first element of the graft is the node v_6 and the leaf of the last element of the graft is v_8 .

As explained in Paragraph 6.2.2.2, the LeAG syntax for Schema-based and Simulation-based multilayering is different. In Schema-based multilayering, the opening colour tag must occur within the ToT that defines the first element of the graft; symmetrically,

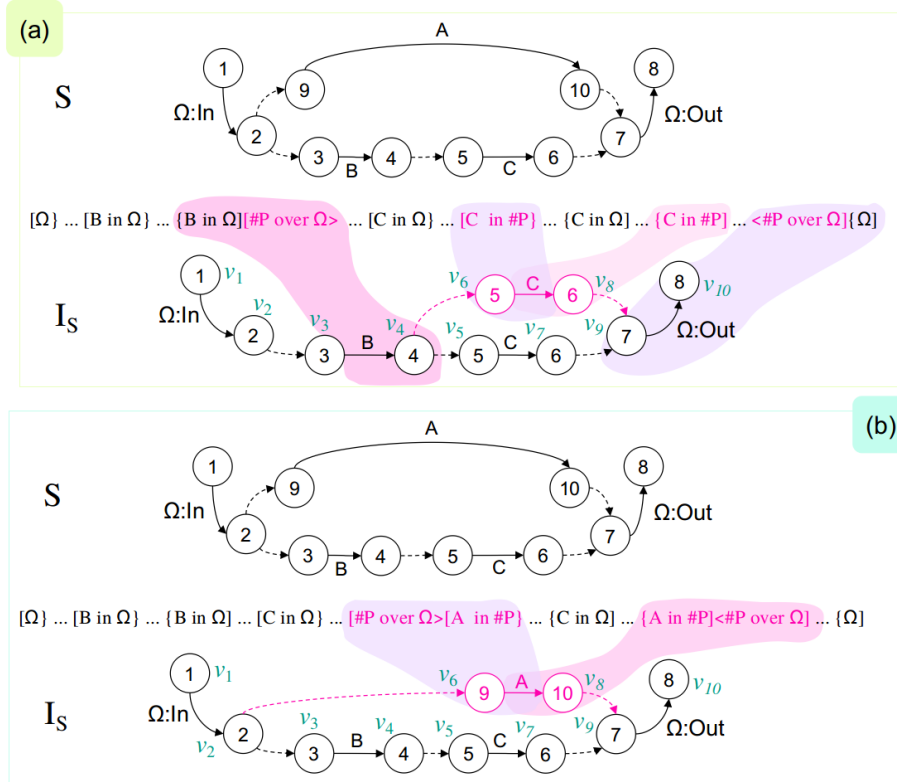


Figure 7.1: Two corresponding SeAG/LeAG/eAG triples, illustrating (a) Simulation-based multilayering and (b) Schema-based multilayering. The coloured areas highlight the correspondence between the ToTs and the elements they define.

the closing colour tag must occur within the ToT that defines the last element of the graft – see Figure 7.1.(b) page 176. By contrast, in Simulation-based multilayering, the opening colour tag must occur in the ToT that defines the starting node of the graft, while the closing colour tag must occur in the ToT that defines the ending node of the graft – see Figure 7.1.(a).

As explained in Paragraph 6.2.2.2, sometimes the starting node of a graft and the root of the first element relate to the same ToT (i.e. have the same reference value), and symmetrically for the leaf of the last element and the end of the graft. The parsing strategy for those settings have already been described in Paragraphs 7.2.1 and 7.2.2 – we briefly re-introduce this strategy in more details in Paragraph 7.2.5.1. In those situations, Schema-based and Simulation-based multilayering are indistinguishable, syntactically speaking.

More interesting – and more general – is the opposite situation, in which the starting node (resp. ending node) of a graft and the root of the first (resp. the leaf of the last) element of the graft *do not relate to the same ToT*. In such a setting, Schema-based and Simulation-based multilayering are expressed very differently, in a way that demands differential parsing strategies. Those are presented in Paragraph 7.2.5.2 below.

7.2.5.1 The Bordering Nodes of a Graft and its Extreme Elements Occur at the Same Position

We consider here the situation in which the starting node of a graft occurs at the same position as the root of the first element of that tag. This case divides into two situations: either the two nodes are actually one same node, or not. Since the algorithmic elements that permit to parse the ToTs that match this situation have already been introduced in the previous paragraphs, this part works like a summary. The symmetric situation, that is: the ending node of the graft and the leaf of its last element occur at the same position, works the same exactly.

The first root and the starting node are the same node. Consider a schema S , a LeAG L containing a graft. Let us distinguish between the **context element** and the **graft hierarchy**, the first being the element (and its constituting sub-elements) the graft is defined over and the second constituting the graft. Let us denote φ_e the path from the schema that corresponds to the context element and φ_g the path describing the graft hierarchy. Necessarily, there are two nodes r_g, l_g from φ_e that serve as the root and leaf for φ_g .

Be X the first element of the graft and X_S the subgraph of S corresponding to that element. The situation we consider here is characterized by $root(X_S) = r_g$.

From there on, many situations may correspond to the above condition. In Figure 7.2 page 178, we illustrate the four possible situations corresponding to the ToT $\{A \text{ in } X\} \{ \#R \text{ over } X \} \{ B \text{ in } \#R \} \{ C \text{ in } X \}$. The key idea is to consider that, when the colour tag $\{ \# R \text{ over } X \}$ belongs to the ToT, the root of an element whose context is $\#R$ belongs to the same hierarchy as the leaf of an element whose context is X . Based on those considerations, we simply apply the parsing algorithms described in Paragraphs 7.2.1, 7.2.2 and 7.2.3 – the latter only if one¹¹ of the elements surrounding

¹¹Note that no two elements in this setting can be inserts. Cf. Property 7.2.3.2.

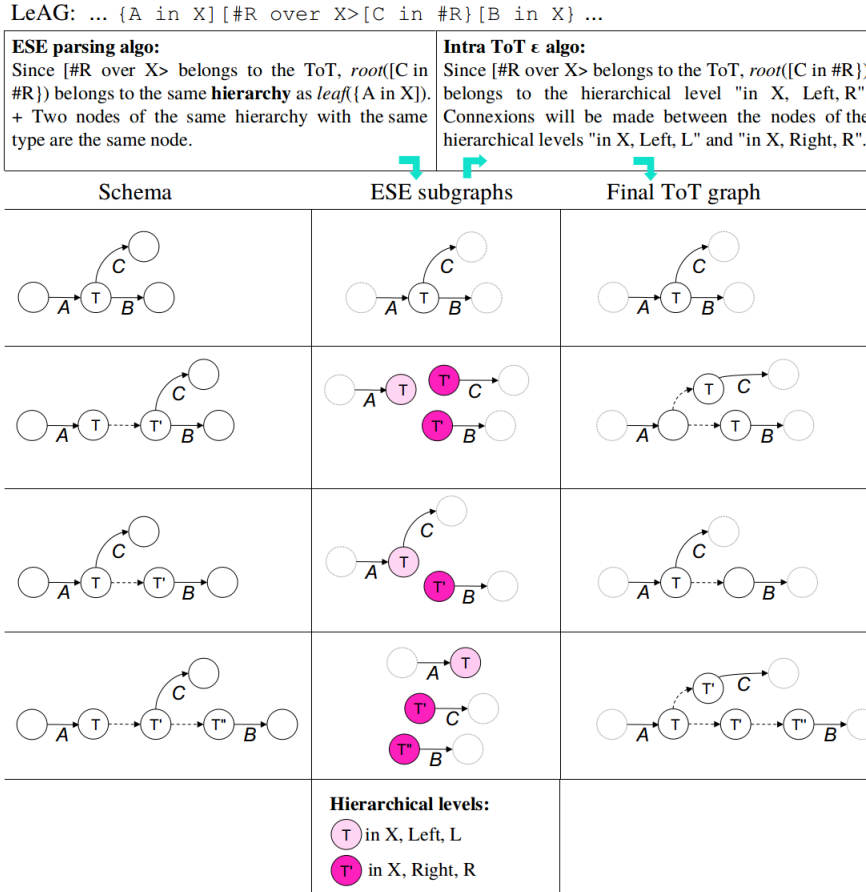


Figure 7.2: Parsing strategy for the LeAG $\{A \text{ in } X\} [\#R \text{ over } X>[C \text{ in } \#R}\{B \text{ in } X\}$, with schemas so that the starting node of the graft and the root of its first element are one same node.

the derivation of \wp_g away from \wp_e is an insert.

One can also check that the same procedure works just as fine for the symmetric situation, in which the ending node of the graft and the leaf of the last element of the insert share the same reference value.

The last leaf and the ending node are the same node. The exact same procedure works here also, as illustrated on Figure 7.3, which rests upon the following ToT: $[A \text{ in } X] [\#R \text{ over } X>[C \text{ in } \#R}\{B \text{ in } X\}$.

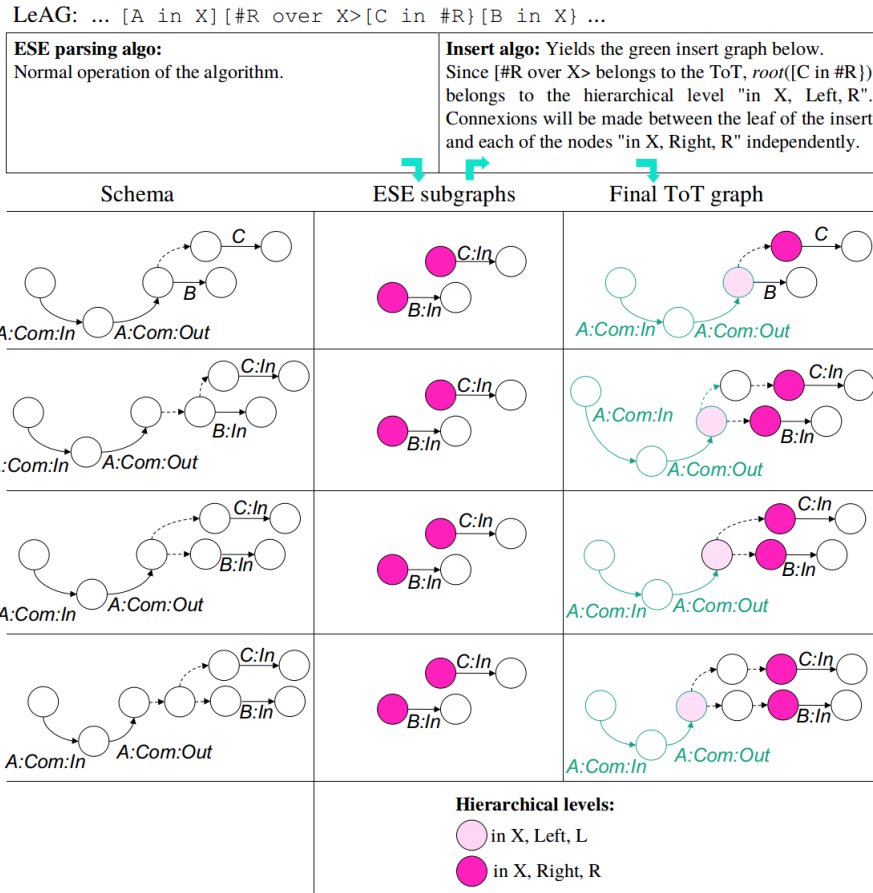
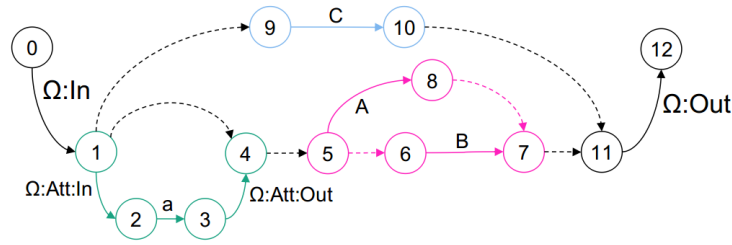


Figure 7.3: Parsing strategy for the LeAG [A in X] [#R over X>[C in #R}[B in X], with schemas so that the starting node of the graft and the root of its first element are not the same node.

7.2.5.2 The Bordering Nodes of a Graft and its Extreme Elements Occur at Different Positions

We now consider here the situation in which the starting node of a graft occurs at a different position than the root of the first element of that tag and thus, result from different ToTs. As we have seen, this case divides into two cases: Schema-based and Simulation-based multilayering.

In order to illustrate both homogeneously, let us consider the following schema. It defines one element Ω , containing either an ESE element C or an optional attribute element (depicted in green below – the content of the attribute is limited to one element a for the sake of simplicity), followed by two alternative ESE elements A or B .



Because of the Simulation-based multilayering abilities of LeAG, one can write an annotation in which two elements C , or an element A and an element B , for instance, overlap freely – or any elements, either of the same name or of different names.

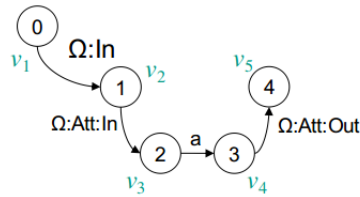
Since the schema also contains three parallel paths, annotation is open to Schema-based multilayering. Here, the schema is quite particular. Indeed, it makes use of the special pattern introduced in Paragraph 6.2.2.2 for A and B (highlighted in pink in the representation of S above), that enables to control the way those two elements, in a Schema-based multilayer setting, overlap. Here, the elements A and B can overlap, but A cannot start after the beginning of B and cannot end before the end of B .

Simulation-based multilayering In Simulation-based multilayering, the opening colour tag occurs in the ToT i so that the starting node v of the graft belongs to G_i . The advantage is that it is clear, while parsing that ToT i , that a coloured element will occur in some following ToT $i + n$, $n \geq 1$ – and that ϵ edges will be required to connect v to the (undiscovered, so far) root v' of the first element of the graft, in G_{i+n} .

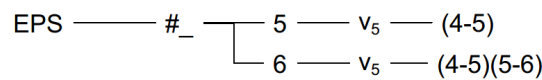
Example. Consider the start of the following LeAG and the schema shown on Figure 7.2.5.2:

$$[\Omega][\text{Att of } \Omega; [a] \dots \{a\}] [\#R \text{ over } \Omega > \dots$$

According to the previous paragraphs, parsing this ToT yields the following graph:



The graph initiates an uncoloured hierarchy. That hierarchy will be completed in a later ToT: by anticipation, according to the strategy given in Paragraph 7.2.4.2, the possible ϵ edges that may connect the leaf of that graph to the next element of the uncoloured hierarchy are the following:



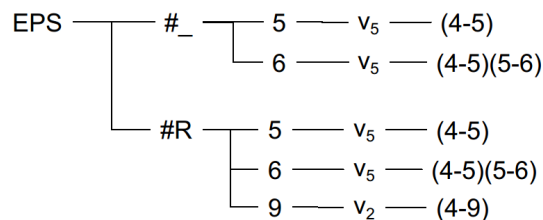
Also, the presence of the tag $[\#R \text{ over } \Omega >$ in the ToT indicates that there is a path deriving from the uncoloured hierarchy leading to the root of the first element of the $\#R$ graft, that will be defined in an upcoming ToT. It also implies that the divergence will occur on a node belonging to the hierarchical level “in X, Right”.

Yet no more can be assessed so far: if the upcoming element is C , then according to the schema, the only possible starting point for the graft shall be v_2 , which is the end of the edge labelled $\Omega: \text{In}$; else, it theoretically could be either v_2 or v_5 .

Thus, the identity of the starting node of the graft as well as the nature of the ϵ sequence from the first ToT graph to the root of the first element of the graft will be determined by the content of the next ToT making use of the colour $\#R$.

Since our aim is to design a single pass parser, we want to anticipate on that upcoming ToT, and to keep track of the set of all the possible ϵ paths that may originate from G_1 – among which one ϵ path will be realised. To do so, we simply add the potential ϵ sequences to the EPS tree-shaped structure defined in Paragraph 7.2.4.2 and illustrated above, that works as a memory of all the ϵ paths that may originate in nodes that have been created, sorted by colour and then by the type of the ending node of each ϵ sequence.

Example. For instance, here, the above EPS structure becomes:



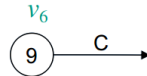
Important remark. Note that in EPS, if two nodes from the same hierarchical level may be the root of two sequences of ϵ edges ending by the same type (e.g. above, v_2 and v_5 can both be connected to a node typed 6, according to the schema), then only the sequences originating in the latter of the two nodes along their common hierarchy will be memorized (i.e. v_5 , here).

Also note it is possible to know, and thus to keep track of, the order of the nodes belonging to the same hierarchical level, along a the corresponding hierarchy, without crawling the graph. Indeed, as indicated on Figure 7.4, the situations in which more than one node related to the same ToT belong to the top hierarchical level of an element X are few. From the typology given in that figure, we can deduce that given a set of nodes related to a given ToT, and given two nodes belonging to the same hierarchical level “in X ”, it is possible to assess which of the two nodes will precede the other in the eAG graph, based on the complete value of the hierarchical level of each node. Thus, requiring to know the order of the nodes within one hierarchical level does not require to crawl the graph and is not contradictory with the goal of designing a single pass parser.

Then, consider that the end of the LeAG is the following:

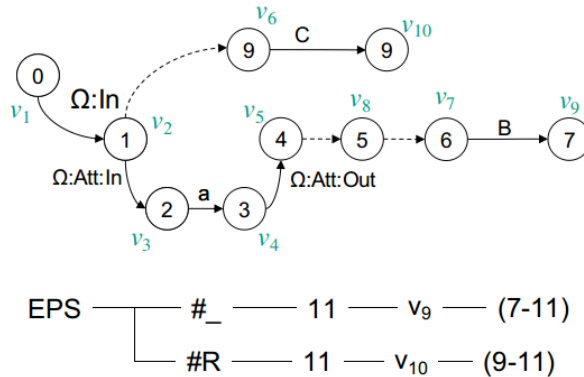
[C in #R} ... [A in Ω } ... {A in Ω } ... C in #R] ... <#R over Ω]{ Ω }

The next ToT contains a tag with #R in its context. Parsing this ToT yields the following graph, made out of an incomplete edge only.



To connect this graph with the previous ToT graphs, one just follows the procedure given in Paragraph 7.2.4.2: EPS/#R/9 gives v_2 , EPS/#R/9/ v_2 gives (4-9), so the ϵ sequence to instantiate is a single edge (ϵ , 4, 9) rooted on v_2 and ending on v_6 .

Parsing the next three ToTs also follows the procedure described in Paragraph 7.2.4.2. So before parsing the last ToT, the graph obtained so far is:



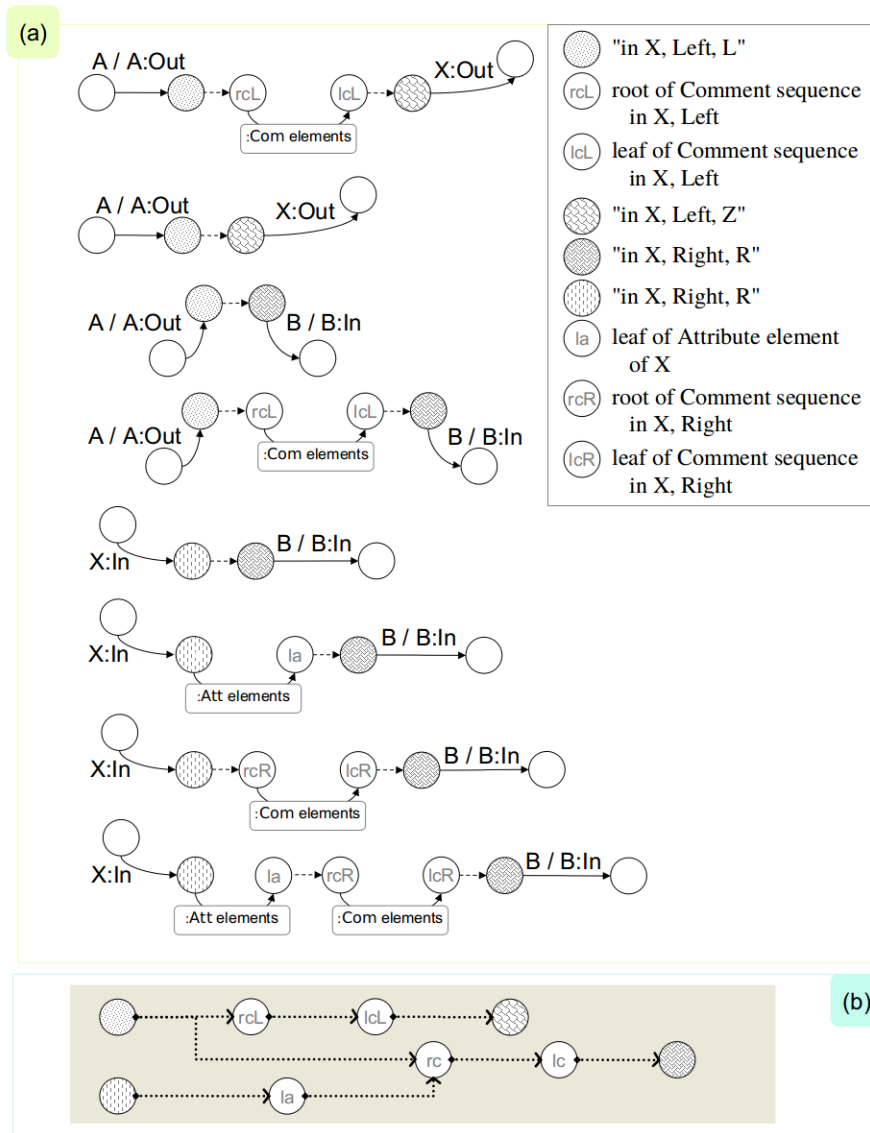
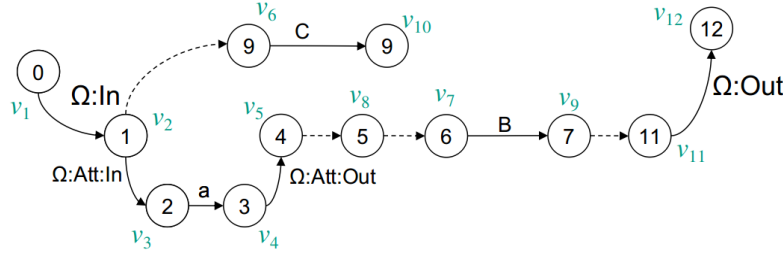
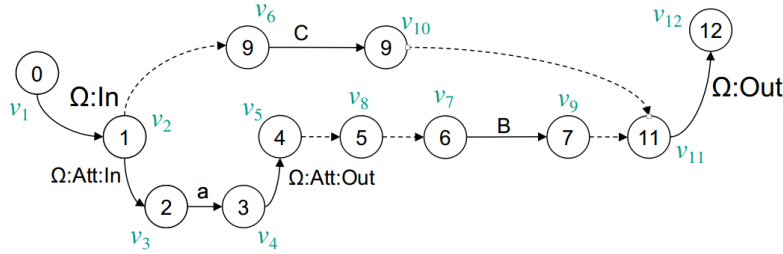


Figure 7.4: (a) Catalogue of the possible situations in which more than one node may belong to the same hierarchical level "in X". (b) The typology above shows that just knowing the complete value of the hierarchical level of two nodes, in the shape "in X, A, B", is enough to assess which of the two nodes will precede the other in the eAG graph. Precedence is represented by the dotted arrows. The precedence relation is transitive.

The remaining ToT is the following: $\langle \#R \text{ over } \Omega \{ \Omega \} \rangle$. The ESE tag is parsed, which yields a single edge labelled $\Omega:\text{Out}$. The root of this edge is connected to the uncoloured hierarchy as indicated in Paragraph 7.2.4.2, once again, which results in:



The presence of a Simulation-based defining ending colour tag is then managed as follows: since $\#R$ is defined over the uncoloured hierarchy, the first node of the current ToT, along that uncoloured hierarchy, that can be connected to according to $\text{EPS}/\#R$, is connected so. Here, $\text{EPS}/\#R$ is reduced to a single value, 11; the first (and only, here) node of a matching type with $\text{EPS}/\#R$ is v_{11} . Thus the epsilon sequence described by $\text{EPS}/\#R/11 = v_{10}$ and $\text{EPS}/\#R/11/v_{10} = (9-11)$, that is, a single ϵ edge between v_{10} and v_{11} , is instantiated. Since no other colour is ending here, and since that ToT is the last of the LeAG, this puts an end to the parsing. The resulting graph is:



This is indeed an eAG that corresponds to the LeAG.

Schema-based multilayering. By contrast, in Schema-based multilayering, the opening colour tag occurs in the ToT k defining the ToT graph G_k so that the root v' of the first element of the graft belongs to G_k . In other words, the opening colour tag **does not** occur in the ToT i , $i < k$, defining the ToT graph G_i so that the starting node of the graft belongs to G_k . Thus there is no indication, in the ToT i , that there will be a derivation from a node of G_i to a node of a new colour. Yet a connection will have to be made between v' the root of the first element of the graft and its starting node v . Two possible strategies can be operated to do so: deduction or anticipation. *Deduction* would mean to parse all the ToTs until the ToT i , and then, when facing the fact there is a Schema-based graft, deduce from the schema what sequence(s) of ϵ

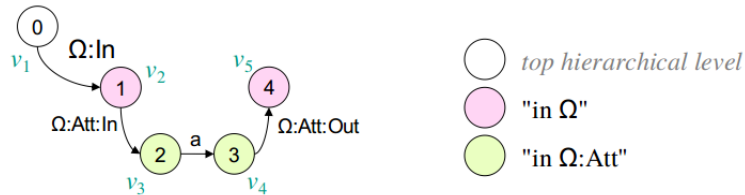
edges may end on v' – and thus find the closest, previously defined node that would both belong to the hierarchy the graft is defined over and have the same type as one of the roots of the ϵ sequences found previously. This strategy shall require back-crawling the graph assembled so far (by connecting together $\{G_i, l < i\}$), which is a time-consuming operation.

Anticipation extends the strategy at work for Simulation-based multilayering. It roughly means initiating all the possible sequences of ϵ edges originating from any node of each ongoing hierarchical level (that is, of nodes involved in elements that have not ended yet, and that may, as such, still work as a context element for an unexpected Schema-based graft). This operation is not costly in time, but in space. Since, following Murata’s assertion that time complexity matters more than space complexity regarding document processing [126], we will favour this option compared with the other. The detail follows.

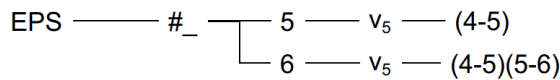
Example. Consider the start of the following LeAG and the schema shown on Figure 7.2.5.2:

[Ω]{Att of Ω ; [a]...{a] }...

According to the previous paragraphs, parsing this ToT yields the following graph, in which the hierarchical level to which the nodes belong to is indicated:



That hierarchy will be completed in a later ToT: by anticipation, according to the strategy given in Paragraph 7.2.4.2, the possible ϵ edges that may connect the leaf of that graph to the next element of the uncoloured hierarchy are the following:

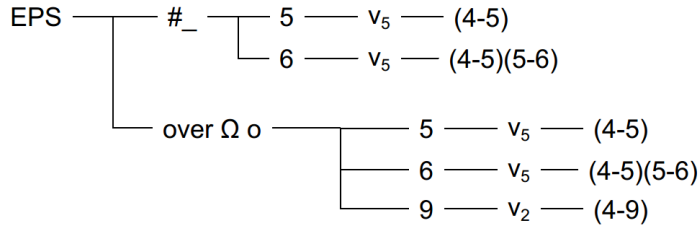


While there is no indication that any of the above nodes will serve as the starting node for a graft, we shall anticipate on that possibility:

Property. For any hierarchical level “in x ” that is opened and not closed in the current ToT, and that is represented in the ToT graph by a node v that can be the origin of a sequence of ϵ edges, there can be a graft whose context is “in x ” whose first element’s root shall be connected to v by that series of ϵ edges.

In particular, in the above ToT graph: we have three hierarchical levels, among which one is ending within the ToT (since it corresponds to the inside of an insert element). The top hierarchical level is represented by a single node typed 0, while no ϵ edge originates in the node of type 0 in the schema (see page 180). The hierarchical level “in Ω ”, on the contrary, is represented by two nodes, v_2 of type 2 and v_5 of type 5 respectively, that each can be the source node of an ϵ sequence and thus, are candidates for being starting nodes for grafts built onto Ω .

We memorize that information in the EPS structure, but in a way that indicates that the ϵ sequences in question are relative to an opening (‘o’) Schema-based layer over Ω :

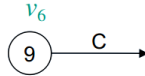


Note that, just as for Simulation-based multilayering, if two nodes from the same hierarchical level may lead to the same nodes by means of ϵ edges, then only the latter node along that hierarchy is to be considered.

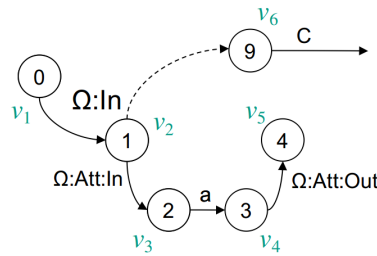
Then, consider that the end of the LeAG is the following:

- (ToT 2 & 3:) [#R over Ω] > [C in #R] ... [A in Ω] ...
- (ToT 4 & 5:) [#G over Ω] > [B in #G] ... {A in Ω } ...
- (ToT 6:) {B in #Blue} < #Blue over Ω] ...
- (ToT 7:) {C in #Red} < #Red over Ω] { Ω }

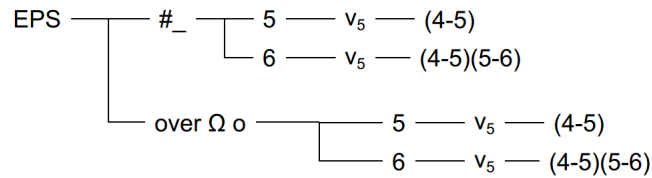
The ToT 2 contains an opening colour tag over Ω , plus one ESE defining tag whose context is that same colour. Parsing the ESE tag yields the following ToT tag:



Then, that tag has to be connected to the previously defined ToT tags. Since its root is typed 9, belongs to the hierarchical level “in #Red” and since the tag [#Red over Ω] belongs to the ToT, the root of the epsilon edges connecting the ToT to the previous one is given by EPS/over Ω o/9 = v_2 , and the ϵ sequence is limited to a single edge ϵ_{49} . This provides the following graph:



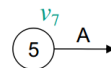
Importantly, the ϵ sequence used to connect the first element of this graft has to be suppressed from EPS, since it cannot be used anymore: else, it would be possible to instantiate a second element C with the Schema-based multilayering syntax, while an element cannot repeat between in two different schema-based layers. So EPS is updated as follows:



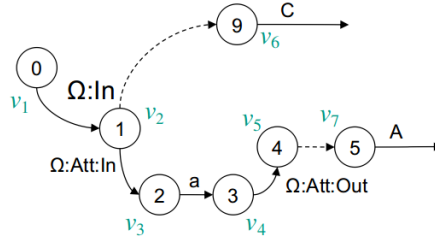
Counter Example 1. Had the ToT 2 been $[\#R \text{ over } \Omega] > [C \text{ in } \#R] \{ \#G \text{ over } \Omega > [C \text{ in } \#G] \}$, the ToT graph would have been made out of two incomplete, parallel C edges, originating in two different nodes both typed 9. Then, $\text{EPS}/\text{over } \Omega \text{ o}/9$ would have been queried in order to determine the ϵ connection for the first node – which would have given the value v_2 (4-9) as above. This value would then have been erased from EPS. $\text{EPS}/\text{over } \Omega \text{ o}/9$ would have been queried once again, giving no result this time – hence resulting in a parsing error.

This mechanism prevents from opening two schema-based grafts where only one was allowed.

The next ToT contains a single ESE defining tag that belongs to the hierarchical level “in Ω ”. Parsing this ToT yields the following graph:



That graph may be connected to the previous ToT graphs, by means of the ϵ edges detailed in $\text{EPS}/\#_ /5 = v_5/(4-5)$. This gives the following, resulting graph:



Yet, very importantly, since the ϵ_{45} edge has been instantiated along the uncoloured hierarchy, then it cannot be instantiated along any schema-defined graft on the uncoloured hierarchy (else, as above, it would be possible to define a schema-based graft containing an element A , which is contradictory with the notion of schema-based multilayering). Thus, the ϵ sequence defined in from “EPS/over Ω o” and **containing** the ϵ edges used to connect the current ToT graph to the previous ones must be deleted. Here, this means that the two remaining values in “EPS/over Ω o” will have to be erased from it.

On the contrary, the ϵ sequences possibly originating in the new nodes from the uncoloured hierarchy and belonging to the hierarchical level “in Ω ” must be added to EPS/over Ω o”.

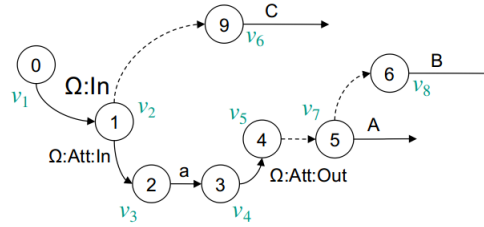
In the end, the new value of EPS is the following:

$$\text{EPS} \text{ ——— over } \Omega \text{ o} \text{ ——— } 6 \text{ ——— } v_7 \text{ ——— (5-6)}$$

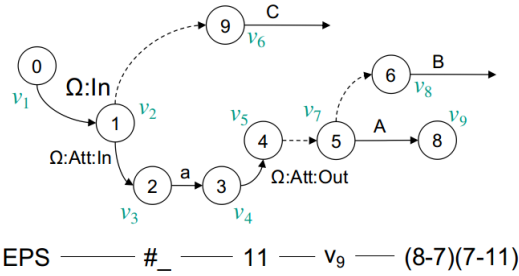
Counter Example. Had the ToTs 3 and 4 been $[B \text{ in } \Omega] \dots [\#G \text{ over } \Omega > [A \text{ in } \#G]]$, then the ϵ sequence used to connect the root of B to the previous ToT graphs would have been (4-5)(5-6). As a consequence, EPS would have been emptied totally. Then, when parsing the next ToT defining the $\#G$ graft, it would have been impossible to connect the root of the first element of that graft to any previous node, since EPS would have been empty.

This mechanism ensures the effectiveness of patterns like the pink one in the schema (cf. Paragraph 7.2.5.2), that aim at controlling the way elements overlap (e.g. here, A cannot start after B started).

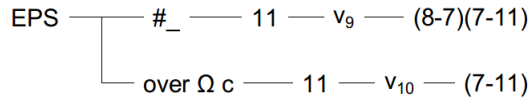
The following ToT is made out of an opening colour tag defining a graft over Ω , and an ESE defining from that graft. Building the ToT graph and connecting it to the previous graphs is done as for the ToT 2. It yields the following graph, while EPS is now empty:



The next ToT is {A in Ω}, which can be parsed as usual. The resulting graph and EPS are:



The next ToT is {B in #Blue}<#Blue over Ω}. Parsing the ESE tag provides a leaf v_{10} of type 7 to the incomplete edge B rooted in v_8 . Importantly, the ToT also contains the closing colour tag for this graft. Then, the leaf of the last element of the graft, namely v_{10} , will be the root for a sequence of ϵ edges whose end will be a node of the hierarchical level “in Ω ”. The schema indicates that the only possible ϵ sequence is made out of the edge $(\epsilon, 7, 11)$. We memorize that information in the EPS structure, but in a way that indicates that the ϵ sequences in question are relative to a closing (‘c’) Schema-based layer over Ω :



Importantly, EPS has to verify the property below.

Property. If, for some context value r , “EPS/over $r\#c$ ” is defined and indicates a certain ϵ sequence e , then it is not possible to add any other epsilon sequence neither in “EPS/over r ” nor in “EPS/# c ” that subsumes e . The reason behind that property is to ensure the effectiveness of patterns like the pink one in the schema (cf. Paragraph 7.2.5.2), that aim at controlling the way elements overlap (e.g. here, A cannot start after B started).

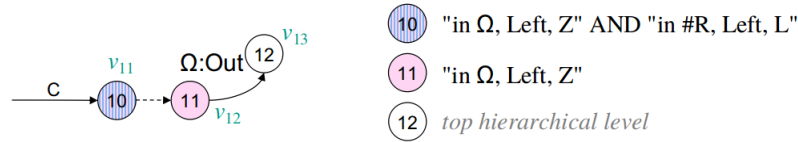
Counter Example 3. Suppose that the ToT 5 and 6 were inverted, that is, suppose that {B in #Blue}<#Blue over Ω} should occur before {A in Ω} in the ToT. Before the patterns are parsed, EPS is empty. Parsing {B in #Blue}<#Blue over Ω} would give the following EPS:

EPS ——— over Ω c ——— 11 — v_{10} — (7-11)

Then parsing $\{A \text{ in } \Omega\}$ yields a graph with a leaf v_l of type 8 belonging to the hierarchical level “in Ω ”. Nodes of that type, according to the schema, can be the root of an ϵ edge made out of two ϵ edge (8-7) and (7-11). Yet it is not possible to include $v_l/(8-7)(7-11)$ in “EPS/#”, because that ϵ sequence subsumes one indicated by “EPS/over Ω c”.

Thus, the annotation pattern, expressed using the schema-based multilayering syntax, aiming at defining an element A ending after an element B belonging to a related graft, and hence contradicting the schema, is made impossible.

The next ToT contains a coloured ESE defining tag $\{C \text{ in } \#R\}$, an ending colour tag $\langle \#R \text{ over } \Omega \rangle$ and an uncoloured ESE tag $\{\Omega\}$. Those three tags are parsed as described in Paragraph 7.2.5.1, which yields the following graph, in which the hierarchical level the nodes belong to are highlighted.

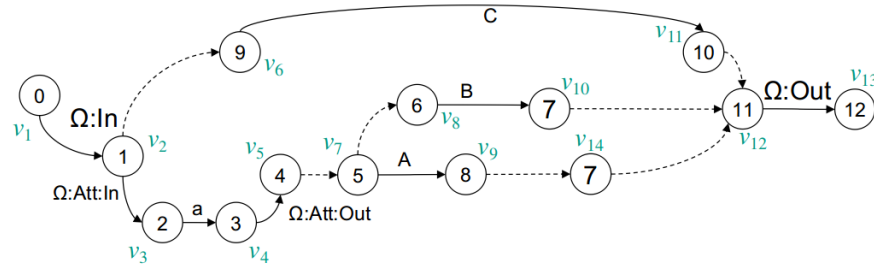


Connecting this graph to the previous ToT graphs works in three steps:

1. the incomplete edge C is completed as defined in Paragraph 7.2.4.1;
2. the first uncoloured node of the ToT, namely v_{12} , has to be connected as indicated in Paragraph 7.2.4.2.
3. last, since the ToT graph contains uncoloured nodes, it must be attempted to connect those to the nodes defined in ‘EPS/over Ω c’. To do that, the uncoloured nodes of the ToT graph must be considered from the earlier to the latter along the uncoloured hierarchy: ‘EPS/over Ω c’ = 11/ v_{10} /(7-11). Thus, the first node typed 11 found along the uncoloured hierarchy must be connected to v_{10} by means of the edge (ϵ , 7, 11).

The match for this condition is v_{12} .

Hence the following graph, which is a fine corresponding eAG for the LeAG, according to the schema S :



7.3 Parsing Algorithm

7.3.1 Data Structures

In the following, data structures' names are boxed.

When a structure $\boxed{\text{dic}}$ is a dictionary, then $\boxed{\text{dic}}.k$ denotes the value associated to the key k .

When a structure $\boxed{\text{tree}}$ is a hierarchy whose leaves contain values and whose nodes bear a unique name $l1, l2, l1.1, l1.2$, etc., then $\boxed{\text{tree}}/l1/l1.1$ denotes the subtree rooted in $l1.1$.

When a structure $\boxed{\text{list}}$ is an ordered set, then $\boxed{\text{list}}_k$ denotes the k^{th} element of that structure.

The data structures needed to parse a LeAG are:

1. Perennial structures

(a) $\boxed{\text{S}}$

Role: Stores the schema, i.e. the LTT' triples allowed by the schema.

Nature: Dictionary with the labels as keys and (T, T') couples as values.

Describing rule: $\boxed{\text{S}}.l = (t, t')$ iff there is an edge labelled l originating in the node of type t and ending on the node of type t' in the schema.

(b) $\boxed{\epsilon^*}$

Role: Provides a quick access to the ϵ sequences allowed by the schema.

Nature: Dictionary with type values as keys, and a list of pairs (T', seq) as values, where T' is a type value and seq a sequence of types describing an ϵ sequence.

Describing rule: $(t', [t_1 \dots t_n; t']) \in \boxed{\epsilon^*}.t$ iff the shortest¹² ϵ sequence between the node typed t and the node t' from the schema goes through the nodes typed $[t_1 \dots t_n]$.

(c) $\boxed{\text{E}}$

Role: Stores the edges of the eAG corresponding to the input LeAG.

Nature: Dictionary with the LTT' triples as keys, and a list of node identifier couples as values.

¹²The unicity of a shortest sequence is guaranteed by restriction (CE1) – see page 156.

Describing rule: $(v_r, v_l) \in \boxed{E_0}.l t t'$ **iff** there is an edge labelled l originating in the node (of identifier) v_r of type t and ending on the node v_l of type t' in the schema.

(d) \boxed{V}

Role: Stores the characteristics of the nodes of the eAG.

Nature: Dictionary with (machine) node identifiers as keys, and a list of four fields: *type*, *reference*, a set of *colours* and optionally *ID* (user defined identifier).

Describing rule: $\boxed{V}.id = [t, r, \{\#c_1, \dots, \#c_n\}]$ **iff** the node of identifier id is typed t , is associated with the reference value r , is related to tags coloured $\#c_1 \dots \#c_n$, and is not associated to any user defined ID value.

(e) \boxed{Tgt}

Role: Stores the nodes for which there is an user defined identifier, together with a binary value that is true if the node was used as a non-target node in the LeAG.

Nature: Dictionary with user-defined node identifiers as keys and a pair of a node identifier and a binary number as value.

Describing rule: For a given Id , $\boxed{Tgt}.Id$ is defined **iff** there is a tag t so that i is a value that appears in an ID field of t .

Moreover, $\boxed{Tgt}.Id = (i, 0)$ **iff** for any tag t so that Id is a value that appears in an ID field of t :

- i. EITHER t is a link insert and $\exists k \mid t.ID = k \rightarrow Id$ and the leaf of the link element is the node of machine identifier id .
- ii. OR t is a quote insert and $\exists k \mid t.ID_2 = k \rightarrow Id \vee t.ID_2 = i \rightarrow k$ and the root or the leaf, respectively, of the link element is the node of machine identifier id .

2. Buffer and ephemeral data structures

A LeAG document may contain inserts that, in turn, contain LeAG fields. In the parsing algorithm, LeAG fields will be parsed independantly from their surrounding LeAG. Thus, we face a hierarchy of LeAGs.

The LeAG document can be considered as the LeAG of level 0. A LeAG field belonging to an insert that is directly included in an element from the level 0 is considered of level 1, and a LeAG field belonging to an insert directly included in an element from level $n \in \mathbb{N}$ is considered of level $n + 1$.

The following structures are needed for the parsing of a given level. Thus, in practice, the following structures will be declined by level. Be a structure \boxed{Str} . By convention, \boxed{Str} will be relative to the level 0 and \boxed{Str}_n will be relative to any deeper level $n \geq 1$.

The definition of the structures is the following:

(a) $\boxed{\text{OCl}}$

Role: Stores the names of the colours for which an opening colour tag has been found but no closing colour tag. Each colour is associated with a value ‘sim’, ‘sch’ or ‘indif’, based on the type of multilayering the beginning (opening colour node + first coloured tag – see Paragraph 7.2.5.2) of the graft implies.

Nature: List of pairs of colour names and string value ‘sim’, ‘sch’ or ‘indif’.

Describing rule $(\#c, \text{‘sim’}) \in \boxed{\text{OCl}}$ **iff** the opening colour tag of $\#c$ occurs in a ToT that precedes the ToT in which the first element coloured $\#c$ occurs, and the closing tag of $\#cc$ has not been parsed yet.

$(\#c, \text{‘sch’}) \in \boxed{\text{OCl}}$ **iff** the opening colour tag of $\#c$ occurs in the same ToT as the first tag coloured $\#c$ but the first node of the first element coloured $\#c$ is not the starting node of the insert, and the closing tag of $\#cc$ has not been parsed yet.

$(\#c, \text{‘sch’}) \in \boxed{\text{OCl}}$ **iff** the opening colour has been parsed and the closing tag of $\#cc$ has not been parsed yet, and none of the above situation was verified.

(b) $\boxed{\text{OECt}}$

Role: Stores the classes of equivalent contexts due to the presence of *opening* colour tags, within the ToT being parsed.

Nature: Dictionary with context names as keys and context names as values.

Describing rule: $\boxed{\text{OECt}}.r = x$ **iff:**

- i. $\exists \#c$ so that $r = \#c$ AND $[\#c \text{ over } x >$ belongs to the current ToT AND there is an element-defining tag t in the current ToT so that $\#c \in t.colours$,
- ii. OR alternatively: $(\forall \#c, r \neq \#c)$ AND $r = x$ AND $\exists \#c_0$ so that $\boxed{\text{OECt}}.\#c_0 = r$.

(c) $\boxed{\text{CECt}}$

Role: Stores the classes of equivalent contexts due to the presence of *closing* colour tags, within the ToT being parsed.

Nature: Dictionary with context names as keys and context names as values.

Describing rule: $\boxed{\text{CECt}}.r = x$ **iff:**

- i. $\exists \#c$ so that $r = \#c$ AND $<\#c \text{ over } x]$ belongs to the current ToT AND there is an element-defining tag t in the current ToT so that $\#c \in t.colours$,
- ii. OR alternatively: $(\forall \#c, r \neq \#c)$ AND $r = x$ AND $\exists \#c_0$ so that $\boxed{\text{CECt}}.\#c_0 = r$.

(d) Structures that partition the set of tags of the ToT:

i. ESE

Role: Stores the ESE defining tags, ranked by ascending¹³ number of colours.

Nature: List of tuples $(o/c, name, colours, context, ID)$, where *colours* and *context* are lists of radicals.

ii. Att

Role: Stores the attribute insert tags of the ToT.

Nature: Dictionary with context values as keys and a list of pairs $(ID, LeAG)$.

iii. InsR

Role: Stores the comment insert tags occurring after the last non-insert tag of the ToT.

Nature: Dictionary with context values as keys and a list of quadruple $(name, colours, ID, LeAG)$, where *colours* and *context* are lists of radicals.

iv. InsL

Role: Stores the remaining comment insert tags.

Nature: Dictionary with context values as keys and a list of tuples $(name, colours, ID, LeAG)$, where *colours* and *context* are lists of radicals.

v. Link

Role: Stores the link insert tags of the ToT.

Nature: List of tuples $(name, colours, context, ID, LeAG)$.

vi. Void

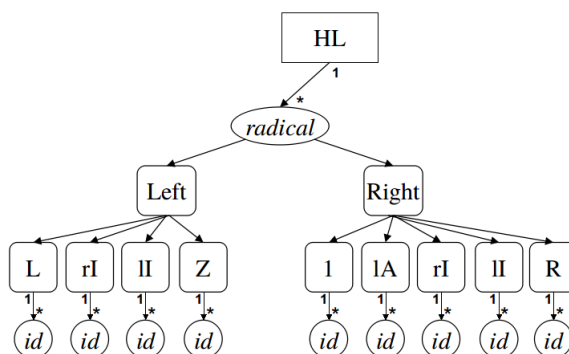
Role: Stores the void insert tags of the ToT.

Nature: List of pairs $(context, ID)$.

(e) HL

Role: Memory of the hierarchical level each node belongs to.

Nature: Hierarchical structure, as follows:



¹³The list is meant to be read from the last to the first element, i.e. in descending number of colours.

Italics denote the places where values can be inserted. Square nodes are constant nodes. The names of the constants mimic the typology of hierarchical levels shown on Figure 7.4 page 183.

Example: $\boxed{\text{HL}}/r/\text{Left}/L = v$ iff $\exists n$ so that there is a tag $\{n \text{ in } r\}$ in the current ToT, and v is the leaf of the element associated with that ESE tag.

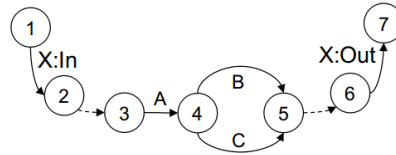
(f) $\boxed{\text{wt}\vdash}$ & $\boxed{\text{wt}\dashv}$

Role: $\boxed{\text{wt}\vdash}$ keeps track of the colours of the incoming edges that do not occur among the colours of the outgoing edges of the nodes of the ToT graph.

$\boxed{\text{wt}\dashv}$ keeps track of the colours of the outgoing edges that do not occur among the colours of the incoming edges of the nodes of the ToT graph.

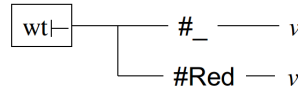
Nature: Dictionaries with colours as keys and node identifiers as values.

Example: Consider the following ToT and schema.

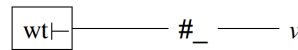


... {A in X, #Red} [B in #Red] [C in X] ...

Parsing the first tag will give one node v , serving as the leaf for an edge which can be affected two colours: # and #Red. Since v has no outgoing edge at the moment, the missing, outgoing colours # and #Red. Thus the following content for $\boxed{\text{wt}\vdash}$:



Then the second tag is parsed. The node v is used as the root of B , which is a #Red tag. Thus the following update for $\boxed{\text{wt}\vdash}$:

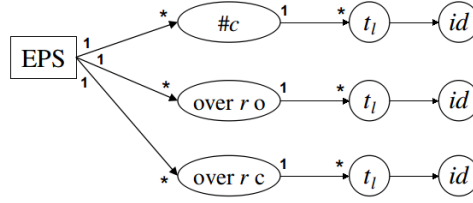


After the last tag is parsed, in this example, $\boxed{\text{wt}\vdash}$ is empty.

(g) $\boxed{\text{EPS}}$

Role: Stores the potential sequences of ϵ edges that originate in nodes resulting from parsed ToTs, and that may connect nodes from further ToTs.

Nature: Hierarchical structure, as follows:



Describing rule: See Paragraphs 7.2.4.2 and 7.2.5.2.

7.3.2 Parsing Algorithm

7.3.2.1 Main Algorithm

Inputs The algorithm takes the following as inputs:

1. A schema \boxed{S} and its corresponding $\boxed{\epsilon^*}$;
2. A LeAG L made out of several ToTs $\{ToT_i\}_{i \in I}$;
3. An integer value deg , denoting the level of the LeAG.

Outputs The algorithm, if it runs normally, yields the following outputs:

- if $deg = 0$:
 1. An eAG made out of the edges \boxed{E} and the nodes \boxed{V} ;
 2. The root and leaf of the eAG.
- if $deg > 0$:
 1. A graph made out of the edges $\boxed{E_{deg}}$ and the nodes $\boxed{V_{deg}}$;
 2. The root and leaf of the top hierarchical level of the graph, i.e. the non-target nodes with an in-degree and out-degree equal to zero respectively.

Main algorithm Initially, \boxed{E} and \boxed{V} are empty, \boxed{S} and $\boxed{\epsilon^*}$ match the user-defined schema and $deg = 0$.

The algorithm runs as follows:

1. Create the ephemeral, empty structures corresponding to the deg attribute.
2. Take each ToT of L in the order of the document. Let ToT^* denote the ToT under consideration.
 - (a) Empty all ephemeral structures but \boxed{OCl} and \boxed{EPS} .
 - (b) Preprocess ToT^* :
 - i. Fill \boxed{ESE} , \boxed{Att} , \boxed{InsR} , \boxed{InsL} , \boxed{Link} and \boxed{void} .
 - ii. Add $(\#c, 'sim')$ to \boxed{OCl} the colours $\#c$ of the opening colour tags for which no element-defining tag of ToT^* has the colour $\#c$.

- iii. If associated with the value 'sim' in $\boxed{\text{OCl}}$, delete from $\boxed{\text{OCl}}$ the colours $\#c$ of the closing colour tags for which no element-defining tag of ToT^* has the colour $\#c$. Else: parsing error.
 - iv. Fill $\boxed{\text{OEct}}$ and $\boxed{\text{CEct}}$.
- (c) For each element Tag^* of $\boxed{\text{ESE}}$, in descending number of colours:
- i. Find the LTT' value associated with $Tag^*/name$ in $\boxed{\text{S}}$.
 - ii. Associate the right identifier either with the root, the leaf, or to both the root and the leaf of the edge associated with Tag^* [see Algorithm 7.3.2.2].
 - iii. If both a root id_r and a leaf id_l were associated with Tag^* :
 - A. insert (id_r, id_l) into $\boxed{\text{E}}.LTT'$;
 - B. $\forall \#C \in Tag^*/colours$, delete $\#C/id_r$ in $\boxed{\text{wt}^-}$;
 - C. $\forall \#C \in Tag^*/colours \setminus \{\#c; \boxed{\text{wt}^-}. \#c = id_r\}$, insert $\#C/id_r$ in $\boxed{\text{wt}^-}$;
 - D. apply 2(c)iiiB and 2(c)iiiC symmetrically to id_l .
 - iv. Else if only a node (root or leaf) id was associated with Tag^* :
 - A. check if there is an incomplete edge from $\boxed{\text{E}}.LTT'$, made out of a node compatible with id , as defined in Table 7.1:
 - α . if there is such an edge, complement it with id .
 - β . else create a new, incomplete edge with id as its only node.
 Let e^* denote resulting edge.
 - B. if $root(e^*)$ is defined, $\forall \#C \in Tag^*/colours$, delete $\#C/root(e)$ in $\boxed{\text{wt}^-}$;
 - C. if $root(e^*)$ is defined, $\forall \#C \in Tag^*/colours \setminus \{\#c; \boxed{\text{wt}^-}. \#c = root(e^*)\}$, insert $\#C/root(e^*)$ in $\boxed{\text{wt}^-}$;
 - D. apply 2(c)ivB and 2(c)ivC symmetrically to $leaf(e^*)$.
- (d) For each (Tag^*, r^*) so that of $\boxed{\text{Att}}.r^* = Tag^*$:
- i. Find the $L_r T_r T'_r$ triple associated with $r^*.name + \text{Att} : \text{In}$ in $\boxed{\text{S}}$.
If the type of $\boxed{\text{HL}}/r^*/\text{Right}/1$ is not equal to T_r : parsing error.
 - ii. Parse the insert Tag^* according to the Algorithm 7.3.2.4, with $\boxed{\text{HL}}/r^*/\text{Right}/1$ as a root candidate and $\boxed{\text{HL}}/r^*/\text{Right}/R$ as a leaf candidate.
Be v^l the leaf returned by the algorithm.
 - iii. Insert v^l in $\boxed{\text{HL}}.r^*/\text{Right}/1A$.
- (e) For each r^* so that $\boxed{\text{InsL}}.r^* \neq \emptyset$:
- i. Set a counter k to zero.
 - ii. For all tag $Tag^* \in \boxed{\text{InsL}}.r^*$:
 - A. Parse Tag^* according to the Algorithm 7.3.2.4, with:
 - if Tag^* is the first element of $\boxed{\text{InsL}}.r^*$:
 $\boxed{\text{HL}}/r^*/\text{Left}/L$, if defined, as a root candidate, or else no root candidate;
 - if Tag^* is the last element of $\boxed{\text{InsL}}.r^*$:
 $\boxed{\text{HL}}/r^*/\text{Left}/Z$, if defined, as a leaf candidate, or else
 $\boxed{\text{HL}}/r^*/\text{Right}/R$, if defined, as a leaf candidate,
or else no root candidate;

- if Tag^* is neither the first nor the last element of $\boxed{\text{InsL}}.r^*$:
no root or leaf candidate.
 - Be $root_k$ and $leaf_k$ the root and leaf returned by Algorithm 7.3.2.4.
 - B. If $k = 0$, return $root_k$ as the root of the insert chain included in r . Insert $root_0$ in $\boxed{\text{HL}}/r^*/\text{Left}/\text{rI}$ if it is different from the root candidate.
 - C. If $k = |\boxed{\text{InsL}}.r^*| - 1$, return $leaf_k$ as the leaf of the insert chain included in r . Insert $leaf_k$ in $\boxed{\text{HL}}/r^*/\text{Left}/\text{II}$ if it is different from the leaf candidate.
 - D. If $k > 0$: connect $leaf_{k-1}$ to $root_k$ by means of the sequence of ϵ edges seq so that $(type(root_k), seq) \in \boxed{\epsilon^*}.type(leaf_{k-1})$. Create new nodes to connect two ϵ edges of that sequence.
 - E. Increment k .
- (f) For each r^* so that $\boxed{\text{InsR}}.r^* \neq \emptyset$:
- i. Set a counter k to zero.
 - ii. For all tag $Tag^* \in \boxed{\text{InsR}}.r^*$:
 - A. Parse Tag^* according to the Algorithm 7.3.2.4, with:
 - if Tag^* is the first element of $\boxed{\text{InsL}}.r^*$:
 $\boxed{\text{HL}}/r^*/\text{Right}/\text{1}$, if defined, as a root candidate, or else
 $\boxed{\text{HL}}/r^*/\text{Right}/\text{1A}$, if defined, as a root candidate,
or else no root candidate;
 - if Tag^* is the last element of $\boxed{\text{InsL}}.r^*$:
 $\boxed{\text{HL}}/r^*/\text{Right}/\text{R}$, if defined, as a leaf candidate, or else no root candidate;
 - if Tag^* is neither the first nor the last element of $\boxed{\text{InsL}}.r^*$:
no root or leaf candidate.
 - Be $root_k$ and $leaf_k$ the root and leaf returned by Algorithm 7.3.2.4.
 - B. If $k = 0$, return $root_k$ as the root of the insert chain included in r . Insert $root_0$ in $\boxed{\text{HL}}/r^*/\text{Right}/\text{rI}$ if it is different from the root candidate.
 - C. If $k = |\boxed{\text{InsL}}.r^*| - 1$, return $leaf_k$ as the leaf of the insert chain included in r . Insert $leaf_k$ in $\boxed{\text{HL}}/r^*/\text{Right}/\text{II}$ if it is different from the leaf candidate.
 - D. If $k > 0$: connect $leaf_{k-1}$ to $root_k$ by means of the sequence of ϵ edges seq so that $(type(root_k), seq) \in \boxed{\epsilon^*}.type(leaf_{k-1})$. Create new nodes to connect two ϵ edges of that sequence. Add $r^*.colours$ to the nodes' colours.
 - E. Increment k .
- (g) For all tag Tag^* in $\boxed{\text{Link}}$:
- i. Find the $L_r T_r T'_r$ triple associated with $Tag^*.name+:\text{LinkTo}:\text{In}$ in $\boxed{\text{S}}$.
 - ii. Find the node v^r of $\boxed{\text{HL}}.Tag^*.context$ whose type is T_r . If there is none: parsing error.
 - iii. Find the $L_l T_l T'_l$ triple associated with $Tag^*.name+:\text{LinkTo}:\text{Out}$ in $\boxed{\text{S}}$.
 - iv. Associate (or create) a target node as the leaf of the element, as defined in Algorithm 7.3.2.5, with $Tag^*.ID_2$ and T'_l as parameters.
Be v^l the returned node.

- v. Parse the insert Tag^* according to the Algorithm 7.3.2.4, with v^r as a root candidate and v^l as a leaf candidate.
- (h) For all tag Tag^* in $\boxed{\text{Void}}$:
 - i. Find the node v of $\boxed{\text{HL}}$. $Tag^*.context/Right/1$. If there is none: parsing error.
 - ii. If $\boxed{\text{V}}$. v/ID is compatible with $Tag^*.ID$:
 - A. Set $\boxed{\text{V}}$. v/ID to $Tag^*.ID$;
 - B. Set $\boxed{\text{Tgt}}$. $Tag^*.ID$ to $(v, 1)$.
- (i) To make the necessary ϵ connections inside ToT^* :

For all r^* so that $\boxed{\text{HL}}$. $r^* \neq \emptyset$:

 - i. Connect the connectible pairs of nodes (v_a, v_b) of the same HL as defined in Algorithm 7.3.2.3, if they have not been connected one to the other so far, by means of the sequence of ϵ edges seq so that $(type(v_b), seq) \in \boxed{\epsilon^*}$. $type(v_b)$. Create new nodes to connect two ϵ edges of that sequence. Add $r^*.colours$ to the nodes' colours.
 - ii. For all $\#c \in r^*.colour$:
 - A. delete $\boxed{\text{wt}\downarrow}$. $\#c.v_a$.
 - B. delete $\boxed{\text{wt}\downarrow}$. $\#c.v_b$.
- (j) To make the ϵ connections with the previous ToTs, apart from nodes belonging to ended colours:

For all remaining coloured nodes $(\#c/v')$ of $\boxed{\text{wt}\downarrow}$:

 - i. find $t = \boxed{\text{V}}$. $v'/type$.
 - ii. If $\exists r$ so that $\boxed{\text{OEct}}$. $\#c = r$,
 - A. if $\#c$ is a starting colour whose starting node is in the ToT, i.e. if $r.colours \subset \boxed{\text{V}}$. $v'/colours$:
 - α . delete $\boxed{\text{wt}\downarrow}$. $\#c/v'$;
 - β . insert $(\#c, 'indif')$ in $\boxed{\text{OCI}}$.
 - B. else:
 - α . find the corresponding node v so that $\boxed{\text{EPS}}$. $'over r o'/t=v$;
 - β . insert $(\#c, 'sch')$ in $\boxed{\text{OCI}}$.
 - iii. Else find the corresponding node v so that $\boxed{\text{EPS}}$. $\#c/t=v$.
 - iv. If they have not been connected one to the other yet: connect v to v' by means of the sequence of ϵ edges seq so that $(type(v'), seq) \in \boxed{\epsilon^*}$. $type(v)$. Create new nodes to connect two ϵ edges of that sequence. Add $r^*.colours$ to the nodes' colours.
 - v. Delete $\boxed{\text{EPS}}$. $\#c/t/v$.
- (k) To make the ϵ connections with the colours that have ended in a previous ToT, and to initiate $\boxed{\text{EPS}}$ in the same move:

For all the contexts r s.t. $\boxed{\text{EPS}}$. $'over r c' \neq \emptyset$:

 - i. For all the nodes v' of $\boxed{\text{HL}}$. r in structural order:
 - A. If ToT^* contains the ending tag of a simulation-based graft over r , i.e. if there is a colour $\#c$ so that $\boxed{\text{CEct}}$. $\#c=r$ and $\boxed{\text{HL}}$. $\#c=\emptyset$:

- α . If there is a node v so that $\boxed{\text{EPS}}.\#c/\text{type}(v)=v$:
connect v to v' by means of the sequence of ϵ edges seq so that $(\text{type}(v'), seq) \in \boxed{\epsilon*}.\text{type}(v)$. Create new nodes to connect two ϵ edges of that sequence. Add $r*.colours$ to the nodes' colours.
Make sure step 2(k)iA is skipped for the next nodes of $\boxed{\text{HL}}.r$.
- B. For all schema-based colour that ended in a previous ToT, i.e. for all v so that $\exists t \mid \boxed{\text{EPS}}/'\text{over } r \text{ c}'/t/v$:
- α . If they have not been connected one to the other yet: connect v to v' by means of the sequence of ϵ edges seq so that $(\text{type}(v'), seq) \in \boxed{\epsilon*}.\text{type}(v)$. Create new nodes to connect two ϵ edges of that sequence. Add $r*.colours$ to the nodes' colours.
- β . For each t, v check that none of the edges of seq have been instantiated previously between nodes of colour belonging to $r.colours$ whose reference value is in the range $[\text{ref}(v), \text{ref}(v')]$. Else: parsing error.
- γ . Delete all $\boxed{\text{EPS}}/'\text{over } r \text{ c}'/T/V$ so that the sequence of ϵ edges between $\text{type}(V)$ and T contains any edge from seq .
- C. In anticipation for any starting, schema-based graft:
- α . Find all the T so that $\exists seq \mid \boxed{\epsilon*}.\text{type}(v')=(T, seq)$
- β . Insert T/v' in $\boxed{\text{EPS}}/'\text{over } r \text{ o}'$.
- (l) For all starting, simulation-based graft, i.e. for all $\#c$ so that $\boxed{\text{OEct}}.\#c \neq \emptyset$ and $\boxed{\text{Ocl}}.\#c='sim'$:
- i. α . Find all the T so that $\exists seq \mid \boxed{\epsilon*}.\text{type}(v')=(T, seq)$
- ii. β . Insert all T/v' in $\boxed{\text{EPS}}/'\text{over } r \text{ o}'$.
- (m) For all closing, schema-based graft, i.e. for all $(\#c, r)$ of $\boxed{\text{CEct}}$ so that $\exists v \mid \boxed{\text{wt}\vdash}.\#c=v$:
- i. if $r.colours \subset \boxed{\text{V}}.v.colours$: delete $\boxed{\text{wt}\vdash}.\#c=v$
- ii. else:
- A. check that $\boxed{\text{Ocl}}.\#c \neq 'sim'$. If not: parsing error.
- B. Find all the T so that $\exists seq \mid \boxed{\epsilon*}.\text{type}(v)=(T, seq)$ and insert T/v in $\boxed{\text{EPS}}/'\text{over } r \text{ c}'$.
- (n) For all contexts r so that $\boxed{\text{wt}\vdash}.r \neq \emptyset$:
- i. check that $\exists!v$ so that $\boxed{\text{wt}\vdash}.r=v$.
- ii. Find all the T so that $\exists seq \mid \boxed{\epsilon*}.\text{type}(v)=(T, seq)$
- iii. Insert T/v' in $\boxed{\text{EPS}}/r$.
- (o) Check that for all colour $\#c$ in $\boxed{\text{Ocl}} \setminus (\boxed{\text{OEct}} \cup \boxed{\text{CEct}})$, $\boxed{\text{EPS}}.\#c$ is not undefined. Idem for the default colour $\#_-$.
- (p) If $ToT*$ is the first ToT, check that $|\boxed{\text{wt}\vdash}| = 1$. Return the corresp. node as the root.
3. Check that $|\boxed{\text{wt}\vdash}| = 1$. Return the corresponding node as the leaf.
4. Check that $\forall ID, (\boxed{\text{Tgt}}.ID)_2 \neq 0$.

Node pair	(v_a, v_b)
1.	$\boxed{V}.v_a/colours \cap \boxed{V}.v_b/colours \neq \emptyset$
2.	$\exists \#c, r_b \mid \#c \in \boxed{V}.v_a/colours \wedge \boxed{OECt}.\#c = r_b \wedge r_b \in \boxed{V}.v_b/colours$
3.	$\exists \#c, r_a \mid \#c \in \boxed{V}.v_b/colours \wedge \boxed{CECt}.\#c = r_a \wedge r_a \in \boxed{V}.v_a/colours$

Table 7.1: Conditions for two nodes to be compatible for being part of the same pair in the data structure E.

7.3.2.2 Associate Nodes to Tags

Inputs A tag Tag^* and a binary root/leaf attribute.

Outputs The algorithm yields an id value.

Algorithm The algorithm runs as follows:

1. Read the PRIMARY, and then the SECONDARY cells of \boxed{HL} associated with either the root or the leaf of the edge associated with Tag^* [see Table 7.2 page 202] until an id value i or $i?$ is found, so that $\boxed{V}.i/type$ matches the type T if the node associated with Tag^* that is under consideration is the root, or else T’.
2. If such a value id is found:
 - (a) check the compatibility of $\boxed{V}.id/ID$ with the (optional) ID field value of Tag^* . If incompatible: parsing error; if not, set $\boxed{V}.id/ID$ accordingly and set $\boxed{Tgt}.id$ to $(id, 1)$.
 - (b) add $Tag^*/colours$ to $\boxed{V}.id/colours$;
 - (c) erase the content of all the PRIMARY cells and write ‘ id ’ instead;
 - (d) add ‘ $id?$ ’ in all the SECONDARY cells containing no value without ‘?’.
3. Else:
 - (a) create a new machine identifier id ;
 - (b) set $\boxed{V}.id/ID$ accordingly with Tag^*/ID and set $\boxed{Tgt}.id$ to $(id, 1)$;
 - (c) insert the new node in \boxed{V} with the right type, reference value, colours and ID;
 - (d) erase the content of all the PRIMARY cells and write ‘ id ’ instead;
 - (e) add ‘ $id?$ ’ in all the SECONDARY cells containing no value without ‘?’.

7.3.2.3 Connect the Connectible Nodes of a Hierarchical Level

Inputs The algorithm takes a context name r as inputs.

Outputs The algorithm builds ϵ connections between the connectible pairs of nodes belonging to the hierarchical level “in r ”.

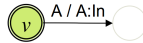
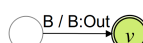
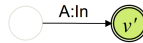
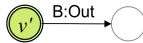
ESE tag Tag^*	Corresponding graph	PRIMARY CELLS of \boxed{HL} for Tag^* (where the green node belongs)	SECONDARY CELLS of \boxed{HL} for Tag^* (compatible with the green node)
$Tag^* = \{A \text{ in } \Pi_i r_i\}$		$\forall i, \boxed{HL} / r_i / \text{Right} / R$	$\forall i$ so that r_i is not a colour: <ul style="list-style-type: none"> $\bullet \boxed{HL} / r_i / \text{Right} / 1$ $\bullet \boxed{HL} / r_i / \text{Left} / L$ $\forall i$ so that r_i is a colour $\#R$ so that $\boxed{OEct} . \#R = r$: <ul style="list-style-type: none"> $\bullet \boxed{HL} / r / \text{Right} / 1$ $\bullet \boxed{HL} / r / \text{Left} / L$ $\forall i$ so that $\boxed{CEct} . r_i = r_i$: <ul style="list-style-type: none"> $\bullet \forall \#c \mid \boxed{CEct} . r_i = \#c,$ $\boxed{HL} / \#c / \text{Left} / L$
$Tag^* = \{B \text{ in } \Pi_i r_i\}$		$\forall i, \boxed{HL} / r_i / \text{Left} / L$	$\forall i$ so that r_i is not a colour: <ul style="list-style-type: none"> $\bullet \boxed{HL} / r_i / \text{Left} / Z$ $\bullet \boxed{HL} / r_i / \text{Right} / R$ $\forall i$ so that r_i is a colour $\#c_i$ so that $\boxed{OEct} . \#c_i = r$: <ul style="list-style-type: none"> $\bullet \boxed{HL} / r / \text{Left} / Z$ $\bullet \boxed{HL} / r / \text{Right} / R$ $\forall i$ so that $\boxed{CEct} . r_i = r_i$: <ul style="list-style-type: none"> $\bullet \forall \#c \mid \boxed{CEct} . r_i = \#c,$ $\boxed{HL} / \#c / \text{Right} / R$
$Tag^* = \{A \text{ in } \Pi_i r_i\},$ $A \in \mathcal{L}_{In}$		$\boxed{HL} / A + Tag^* / \text{colours} / \text{Right} / 1$	<ul style="list-style-type: none"> $\bullet \boxed{HL} / A + Tag^* / \text{colours} / \text{Right} / R$ And if $\boxed{OEct} . A = A$: <ul style="list-style-type: none"> $\bullet \forall \#c \mid \boxed{OEct} . A = \#c,$ $\boxed{HL} / \#c / \text{Right} / R$ $\bullet \boxed{HL} / B + Tag^* / \text{colours} / \text{Left} / L$
$Tag^* = \{B \text{ in } \Pi_i r_i\},$ $B \in \mathcal{L}_{Out}$		$\boxed{HL} / B + Tag^* / \text{colours} / \text{Left} / Z$	And if $\boxed{CEct} . B = B$: <ul style="list-style-type: none"> $\bullet \forall \#c \mid \boxed{CEct} . B = \#c,$ $\boxed{HL} / \#c / \text{Left} / L$

Table 7.2: List of the primary and secondary cells of the structure HL, relative to a given tag.

Algorithm The algorithm runs as follows. Based on the ordered set of items and for each item, of targets, as defined in Figure 7.5:

1. Take $i = 1$.
2. While $i < 9$:
 - (a) If no node is designated by $\boxed{\text{HL}}/r/\text{target}_j$ increment i and go to the start of the loop.
 - (b) If the target field associated with item_i is ‘none’, increment i and go to the start of the loop.
 - (c) Else:
 - i. Find, in the ascending order for j , the first target_j so that $\boxed{\text{HL}}/r/\text{target}_j$ defined.
If none is defined, the algorithm ends.
Else be v the node designated by $\boxed{\text{HL}}/r/\text{item}_i$ and v' the node $\boxed{\text{HL}}/r/\text{target}_j$.
 - ii. If they have not been connected on to the other yet: connect v to v' by means of the sequence of ϵ edges seq so that $(\text{type}(v'), \text{seq}) \in \boxed{\epsilon*}.\text{type}(v)$.
Create new nodes to connect two ϵ edges of that sequence. Add $r*.\text{colours}$ to the nodes’ colours.
 - iii. Set i to j and go to the start of the loop.

7.3.2.4 Insert with LeAG/ID₂ Field Parsing

Inputs The algorithm takes as inputs: a pair of candidate root v_c^r and leaf v_c^l for the eAG element corresponding to the insert and the tag Tag^* .

Output The algorithm builds the element graph. It also yields the ‘root’ and ‘leaf’ of the graph, i.e. the non-target nodes with an in-degree and out-degree equal to zero respectively.

Algorithm The algorithm runs as follows:

1. Find the $L_r T_r T_r'$ triple associated with $r*.\text{name}+\text{Att}:\text{In}$ in $\boxed{\text{S}}$.
2. Find the $L_l T_l T_l'$ triple associated with $r*.\text{name}+\text{Att}:\text{Out}$ in $\boxed{\text{S}}$.
3. If $\boxed{\text{V}}.v_c^r/\text{type} = T_r$:
 - (a) Then if $\boxed{\text{V}}.v^r/\text{ID}$ is compatible with Tag^*/ID_1 :
 - i. Take v_c^r as the ‘real root’ v^r ;
 - ii. Set $\boxed{\text{V}}.v^r/\text{ID}$ to Tag^*/ID_1 .
 - (b) Else: parsing error.
4. Else: create a new node v^r and insert $v^r/T_r/(\text{Tag}^*/\text{colours})/\text{Tag}^*/\text{ID}_1$ into $\boxed{\text{V}}$.
5. Insert $(v^r, 1)$ into $\boxed{\text{Tgt}}.\text{Tag}^*/\text{ID}_1$.

	item			targets, in this order
Left	1	L	to	2 ; 4 ; 9
	2	rI	to	none
	3	II	to	4 ; 9
	4	Z	to	none
Right	5	I	to	none if 6 is defined, else to 7 ; 9
	6	IA	to	7 ; 9
	7	rI	to	none
	8	II	to	9
	9	R	to	none

Figure 7.5: The ordered list of possible *items* in a given hierarchical level (vertical, green) and for each, the ordered list of possible *targets* (horizontal, pink).

6. Apply symmetrically the procedure 3 to 5 to $r * .name + \text{Att:Out}$ and v_c^l . Be v_l the resulting 'real leaf'.
7. If $\exists N, M$ so that $LeAG = N \rightarrow M$:
 - (a) Associate (or create) a target node as the first node of the element, as defined in Algorithm 7.3.2.5, with N and T_r' as parameters.
Be v_{LeAG}^r the returned node.
 - (b) Associate (or create) a target node as the last node of the element, as defined in Algorithm 7.3.2.5, with M and T_l' as parameters.
Be v_{LeAG}^l the returned node.
8. If $LeAG = \emptyset$:
 - (a) Check that $T_r' = T_l'$. If not: parsing error.
 - (b) Create a node of type T_l both as root and leaf output of the algorithm.
9. Else: parse $Tag*/LeAG$ based on \boxed{S} and its corresponding $\boxed{\epsilon*}$, with the degree attribute set to $deg + 1$. Be v_{LeAG}^r and v_{LeAG}^l the root and leaf of the resulting LeAG.
10. Insert (v^r, v_{LeAG}^r) in $\boxed{E} . L_r T_r T_r'$.
11. Delete $\boxed{wt\downarrow} . (r * .colours) . v^r$.
12. Delete $\boxed{wt\downarrow} . \# . v_{LeAG}^r$.
13. $\forall \#C \in (Tag*/colours \setminus \{\#c \in Tag*/context; \boxed{OEct} . \#c \neq \emptyset\}) \setminus \{\#c; \boxed{wt\downarrow} . \#c = v^r\}$, insert $\#C/v^r$ in $\boxed{wt\downarrow}$.
14. Apply symmetrically the procedure 10 to 13 to the node couple (v_{LeAG}^l, v_l) .

7.3.2.5 Target Node Association

Inputs The algorithm takes as inputs: a user-defined identifier Id , a node type T .

Output The algorithm returns a node v_{return} .

Algorithm The algorithm runs as follows:

1. Check if there is a value (v, x) in $\boxed{\text{Tgt}}.Id$.
 - (a) if there is, and if the type of v is not T : parsing error;
 - (b) if there is, and if the type of v is T , choose v as a node v_{return}
 - (c) else:
 - i. create v_{return} . Fill $\boxed{\text{V}}.v_{return}$ accordingly.
 - ii. set $\boxed{\text{Tgt}}.Id$ to $(v_{return}, 0)$.

7.3.3 Parsing Algorithm: Time Complexity

In this paragraph, we qualify the time-complexity of the above algorithm. This evaluation is based upon five natural numbers:

1. n_s , that is the total number of nodes in the schema;
2. e the length of the longest sequence of ϵ edges in the schema;
3. tot , that is the number of ToTs in the LeAG;
4. n , that refers to the number of tags in the whole document¹⁴;
5. p , that is the maximal number of colours that are active at the same time (i.e. the maximal number of stacked layers of annotation in the document).

The estimation of the maximal size of the crucial, following data structures and lists, expressed in terms of the above naturals, is the following:

- For any tag Tag^* , $|\{c \in Tag^* / colours\}| < p$.
- The number of radicals in $\boxed{\text{HL}}$ is inferior to the number of tags in the train of tags $n - tot$.
- The number of colour names c so that $\boxed{wt \vdash}.c \neq \emptyset$ is inferior to p . Given a colour name c , $|\{v \text{ so that } \boxed{wt \vdash}.c.v \neq \emptyset\}| < n - t$.
- Idem for $\boxed{wt \dashv}$.
- The number of colour names c so that $\boxed{\text{EPS}}.c \neq \emptyset$ is inferior to p . The number of contexts r so that $\boxed{\text{EPS}}.\text{'over r o'} \neq \emptyset$ is inferior to n . So is the number of contexts r so that $\boxed{\text{EPS}}.\text{'over r c'} \neq \emptyset$.
The number of type values t in the lists in either $\boxed{\text{EPS}}.c$, $\boxed{\text{EPS}}.\text{'over r o'}$ or $\boxed{\text{EPS}}.\text{'over r c'}$ is inferior to n_s .

¹⁴Thus $n - tot + 1$ is the maximal number of tags in a ToT.

- The cardinal of all the colour sets $\boxed{\text{OCl}}$, $\boxed{\text{OEct}}$ and $\boxed{\text{CEct}}$ is inferior to p .
- For any type value t , the size of $\boxed{\epsilon^*}.t$ is inferior to n_S .

Additionally, the access to any value of a dictionary is $O(1)$. Based on those estimations, that are very pessimistic, we can give the following complexity estimation of the algorithm, bit by bit (when a comment is needed), following the structure of Algorithm 7.3.2.1:

1. The complexity of this bit is independent from the document.
2. The number of iterations of this loop is tot :
 - (a) The complexity of this bit is independent from the document.
 - (b) Filling the structures in i. can be done by reading the ToT from right to left, in a single pass: $\boxed{\text{InsR}}$ can then be filled before $\boxed{\text{InsL}}$, and is fed only until an ESE tag is read. Thus the complexity is $O(n - tot)$.
 - (c) The number of iterations of the loop is inferior to $n - tot$.
 - i. This is done in $O(1)$.
 - ii. This bit requires to run Algorithm 7.3.2.2. The complexity of this algorithm is determined by the complexity of its first step, that is described in Table 7.2. This table states that the primary, and then the secondary cells of $\boxed{\text{HL}}$ have to be read for each context of the tag, until a compatible identifier is found: the number of those accesses to $\boxed{\text{HL}}$ is thus inferior to twice the number of contexts of the tag. Since two contexts of a tag necessarily have a different colour, then the complexity of Algorithm 7.3.2.2 is $O(p)$.
 - iii. This bit requires to read the lists of colours of the tag: this is done in $O(p)$.
 - iv. In A., the list of incomplete edges has to be read. It is worth noting that there cannot be more incomplete edges than there are colours that have been opened and not closed yet. Thus the length of this list, that shall be indexed for fast access, is inferior to p . Then in B. and C. the lists of colours of the tag have to be read. Hence this bit runs in $O(p)$.
 \Rightarrow The section (c) runs in $O([n - tot]p)$.
 - (d) In this section, a tag containing a LeAG field is parsed. Since the parsing of this LeAG field is done independently from parsing the rest of the LeAG, everything happens as if the corresponding LeAG was deported to the end of the document and its ToTs parsed like any other – with the exception that their root and leaf is known. Thus the complexity of this step is equal to the complexity of the instructions that do not correspond to the parsing of the inner LeAG. Here, the complexity for defining the edges surrounding the eAG resulting from the LeAG field is the complexity of the steps i. and iii., which is $O(1)$.
 \Rightarrow The section (c) runs in $O(n - tot)$.

- (e) Idem as above. Here, the complexity for defining the edges surrounding the eAG resulting from the LeAG field is the complexity of the steps i. and ii.B-E., which is $O(1)$.
 \Rightarrow The section (c) runs in $O(n - tot)$.
- (f) Idem as above. Here, the complexity for defining the edges surrounding the eAG resulting from the LeAG field is the complexity of the steps i. and ii.B-E., which is $O(1)$.
 \Rightarrow The section (c) runs in $O(n - tot)$.
- (g) Idem as above. Here, the complexity for defining the edges surrounding the eAG resulting from the LeAG field is the complexity of the steps i., ii, iii. and iv $O(p)$: the complexity of i. and iii. is $O(1)$; the complexity of ii. is $O(p)$ (since the length of the list of contexts is inferior to p); the complexity of iv. has been characterized in step (c) as equal to $O(p)$.
 \Rightarrow The section (c) runs in $O([n - tot]p)$.
- (h) This section is ran for all the void inserts, whose number is inferior to $n - tot$. The operations done for each void insert tag run in constant time.
 \Rightarrow The section (c) runs in $O(n - tot)$.
- (i) This section requires to read all the radicals in $\boxed{\text{HL}}$, whose number is inferior to $n - tot$. For each radical, Algorithm 7.3.2.3 has to be ran. In Algorithm 7.3.2.3, the loop is executed at most 8 times, according to Table 7.5. In the loop, the fact that two nodes have already been connected together or not has to be checked, which can be done in constant time if, whenever an edge is defined, a dictionary whose keys is a concatenation of the identifier of the summit and end of the edge separated by a special character, and whose value is a binary, is maintained. Then, a series of ϵ edges has to be created, which implies at most e edge definitions, which are each done in constant time. Hence the complexity of Algorithm 7.3.2.3 is $O(e)$.
 \Rightarrow Thus the complexity of (i) is $O([n - tot] \cdot e)$.
- (j) This section is iterated for all the remaining nodes in $\boxed{\text{wt}\downarrow}$. At this step of the algorithm, there cannot be more such nodes than there are colours, so the maximal number of iterations is p . Then, at each iteration, the main operation consists in checking, once, if a list of colours $r.colours$ is included in another list $v'/colours$. Both lists contain less than p items. If we make sure that those lists are ordered, this can be checked in $O(p)$. Eventually, at each iteration, a sequence of ϵ edges has to be created, which again can be done in $O(e)$.
Thus the complexity of this step is $O(p \cdot [p + e])$.
- (k) The number of contexts defining an iteration of (k) is inferior is the number of contexts in the ToT, that is itself inferior to $(n - tot)p$. For each of them, 9 nodes have to be considered.
A. Checking the condition A. can be done in $O(p)$. Then α runs in $O(e)$.

The maximal number of iterations of the loop B. is equal to the maximal number of type values in the schema, that is n_s . α runs in $O(e)$. Since we want to identify parallel sequences of ϵ edges, they cannot be more numerous than p , and since $\boxed{\text{E}}$ contains lists describing the edges that can be made so that the last value describes the last edge to have been parsed, β can be done by reading the p last values of the lists in $\boxed{\text{E}}$ defining the sequence of ϵ edges – those lists are at most e . Eventually, γ demands to compare seq and up to n_s lists of up to e elements, which runs in $O(n_s \cdot e^2)$.

\Rightarrow The time complexity for (k) is $O((n - t)p \cdot n_s \cdot e^2)$.

- (l) The maximal number of iterations of this loop equals the number of accesses to $\boxed{\text{OECt}}$, that is inferior to p . Then, for each iteration, the number of accesses to $\boxed{\epsilon^*}$, that is inferior to n_s the total number of type values in the schema.

\Rightarrow The time complexity for (l) is $O(p \cdot n_s)$.

- (m) The maximal number of iterations of this loop equals the number of accesses to $\boxed{\text{CECt}}$, that is inferior to p . Then, in i. checking the inclusion of a colour list into another one can be done, if the lists have been kept in order, in linear time on their length, that is in $O(p)$. In step ii. the number of accesses to $\boxed{\epsilon^*}$, that is inferior to n_s .

\Rightarrow The time complexity for (m) is $O(p \cdot [p + n_s])$.

- (n) This loop iterates for all the contexts in the ToT, that are less numerous than p at this step (after the ϵ edges have been created, there is at most one pending node per colour). Then, the size of the structures that are searched in step i. and ii. is smaller than $(n - tot)$ and n_s respectively.

\Rightarrow The time complexity for (n) is $O(p \cdot [p + n_s])$.

- (o) If the colour lists are ordered, then (o) runs in $O(p)$.

- (p) Step (p) runs in $O(1)$.

\Rightarrow The time complexity for step 2. is $O(tot \cdot p \cdot ([n - tot] \cdot n_s \cdot e^2 + [p + n_s] + [p + e]))$.

3. The time complexity for step 3. is $O(1)$.

4. The time complexity for step 4. is $O(n)$.

It follows that the time complexity of the whole algorithm is $O(n + tot \cdot p \cdot ([n - tot] \cdot n_s \cdot e^2 + [p + n_s] + [p + e]))$. In this expression, we have kept naturals that are not dependent on the document that is parsed but are characteristic of the schema instead: n_s and e . It has yet to be noted that $n_s \cdot e^2$ is a worst case factor, in the sense that n_s , in particular, was used above as an upper bound for the number of type values in lists that will most probably never, in real life, be even close to the total number of type values in the schema. In other words, $n_s \cdot e^2$, which can be quite big (a reasonable value for e would be about 10, and the total number of nodes in a schema could be in the 10^3 order of magnitude), is an extremely pessimistic upper bound for the coefficient factor for the calculation time.

We can also consider that in a real-life document, the maximal number of layers of annotation that may occur simultaneously will hardly go beyond a dozen (handling over 5 simultaneous interlaced layers may even seem unrealistic, from an editorial point of view) and may thus be consider a constant before the much bigger values n and t could be. Indeed, if one considers a very basic annotation of the *Encyclopédie*, in which only the articles would be identified, such an annotation would contain a number of nodes about twice as big as the number of articles in the corpus, that is close to 74.000. Thus, the above expression can be simplified into:

$$O(n + tot \cdot [n - tot])$$

It has to be noted that the values n and tot are not unrelated. It has been stated above that $n - tot + 1$ is the maximal number of tags in a ToT; in other words, $1 \leq tot \leq n$. Let us then consider the second term in the above expression. Then, if tot is close to one, that is, if there are few ToTs in the document, then $tot \cdot [n - tot]$ is in the order of magnitude of n . Now if tot is close to n , that is, if the document contains many small ToT, then $tot \cdot [n - tot]$ is in the order of magnitude of t , i.e. of n .

As a consequence, the time complexity of the above algorithm is linear in terms of the number of tags the document contains.

7.4 Conclusion

In this chapter, we have introduced LeAG, an inline markup syntax for expressing eAG without this requiring any dedicated interface. Several syntactic notions have been introduced in order to make LeAG expressive enough to enable the expression of multilayer annotation containing links, quotes, comments and attributes, like in eAG: the notion of graft, that is a means to define locally an additional annotation layer based on an underlying layer, and the notion of insert, that is made to add critical content inside the document.

We have also defined the algorithm for a schema-aware parser that enables to translate a given LeAG, under some conditions, into an eAG validated by a given schema, if such an eAG exists. The conditional aspect of the parsing, that will fail for LeAG that do not match the given schema, can be used to define the notion of validation for LeAG: a LeAG is valid against a SeAG S iff the result of its parsing, given S , is defined.

Eventually, we have evaluated the time-complexity of this parsing algorithm, which is linear. It is, to the best of our knowledge, the first validation mechanism running in linear time for the validation of cyclic data expressed in an inline markup syntax.

Part IV

Bidirectionalizing eAG/SeAG

Chapter 8

Introduction

8.1 The Problem to Solve

In the Introduction part of this work, we highlighted the needs, in a collaborative editorial setting, 1) for DSE to be grounded on schemas; 2) for schemas to be collaboratively defined; 3) for this collaborative work being incremental and initiated at the individual scale.

We also translated those *métier* needs into technical terms, as follows:

Let us consider that, at a certain moment t , there is a collectively used data structure S . This structure shapes the annotated corpus I_S , that represents the (ongoing) DSE. Then, at a moment $t + \delta$, an editor envisions a different structure for the data, and proposes an alternative data structure S' by modifying S : practically speaking, she defines a transformation on schemas g so that $S' = g.S$.

From there on, the challenge is to assist the production of $I_{S'}$, structured data instantiating S' and sharing ‘as much information’ as possible with I_S , so that the editor shall experiment her proposition of alternative data structure on the widest scale, illustrate her proposition to the other editors and, if this proposition is to be accepted as the new data structure for the whole DSE, so that the editorial team has an instance of this new data structure retaining as much of the previous editorial work as possible, without updating the data by hand (or by means of ad-hoc scripts). Moreover, in order to avoid the situation in which too many editors should work on their own structure without contributing to the collective edition, it should be interesting to have a corresponding translation of the annotations natively expressed in $I_{S'}$ into a shape validated against S .

Hence the need, technically speaking, to derive a bidirectional transformation $Bx = (b_1; b_2)$ so that $b_1.I_S = I_{S'}$ and so that any update in I_S or $I_{S'}$ be propagated to the other domain by either b_1 or b_2 .

$$\begin{array}{ccc} S & \xrightarrow{g} & S' \\ \xleftrightarrow{typ.} & & \xleftrightarrow{typ.} \\ I_S & \xleftrightarrow{Bx} & I_{S'} \end{array}$$

In Part II of this work, we have introduced eAG/SeAG, a schema-aware multistructured data model whose expressive power make it fit for supporting many different editorial projects, including *growing* ones: based on a schema, intuitively, because the model supports multilayering – with self overlap – then if an additional layer of annotation is needed to encode a new annotation paradigm, adding a corresponding layer to the schema and instantiating this layer can be done, regardless of the complexity of the previous annotation. In other words, the graph formalism underlying eAG/SeAG is so unrestrictive that it shall not limit the ‘growth’ of the data structure, compared to other annotation languages, XML first, whose underlying graph formalism is so narrow that any additional structure shall be shoehorned into the existing hierarchy schema – if it can be.

We have also proposed simulation as a validation mechanism. Apart from interesting expressive features (e.g. validation of cyclic graphs, good handling of self overlap, etc.) and good algorithmic properties (simulation-based validation can be checked either on-the-fly or a posteriori, with a square-time complexity), simulation appeared as a promising solution to us precisely in the perspective of managing schema evolution. Indeed, as sketched above, schema evolution demands that a bidirectional transformation on the instances shall be automatically derived from the transformation the editor defines on the schemas. The fact that the eAG/SeAG model rests upon the notion of simulation will, as we will see, help in that derivation. Indeed, contrary to grammar-based validation approaches, in which the validating and the validated objects are of two different natures (e.g. Tree Automaton vs. Tree in XML [126]), eAG and SeAG are both cyclic graphs, sharing the same semantics on labels.

What follows is the presentation of an ongoing work in which we consider how to solve the above problem of schema evolution for eAG/SeAG. This work has not been submitted to peer-review so far. We start by positioning our proposition in the state-of-the art briefly, before presenting our approach.

8.2 Schema Evolution, Bidirectional Transformations

More than a critical, in-depth review of the existing approaches towards the problems of schema evolution and bidirectional transformation, we propose the reader a short panorama of what exists, and why an alternative proposition had to be operated for eAG/SeAG.

The technical problem sketched in the above paragraph can be related to two research problems dealt with rather independently by two different communities: Schema Evolution, in the field of Database studies, and Bidirectional transformations, in the Language community. We provide a rapid overview of both in the following.

8.2.1 Schema Evolution in Database Studies

The problem we want to tackle can be seen as a particular case of the Schema Evolution problem, applied to the context of graph-structured markup data.

Schema Evolution is a variation of the classical Data Exchange (DE) problem [76,

113, 18, 13, 24]. Data Exchange, in general, is the process of taking data structured according to one schema and transforming it into some sensible materialized data validated by another schema, possibly in another model (ex. from an XML file to a relational database), according to a mapping that declares the nature of the correspondence between the source and target schemas. Schema Evolution, in particular, is a Data Exchange situation where both target and source data are expressed in the same model, and where the target schema is obtained by transforming the source schema (e.g. [117] in a slightly different setting) into an updated, naturally related target schema. The general Data Exchange problem has been widely studied in the context of Relational Databases. Recently, due to the multiplication and diversification of exchange settings, alternative contexts came into consideration, including Graph Databases [13, 24] and Semistructured Data [8, 188].

Still, to the best of our knowledge, Data Exchange (and thus Schema Evolution) has not been studied in the context of graph-structured, schema-aware annotation data. Annotation, as mentioned previously, is indeed almost never done “into the wild”: more and more annotation campaigns are collaborative [96, 120, 157]. In order to guarantee a certain consistency across the participants, annotation is often made to rest upon either a schema or an ontology [27, 39, 183, 130].

Schemas and ontologies are two different approaches to modelling. On the one hand, *ontologies* are meant to make explicit a certain conceptualization of a domain, by means of a well documented formalism; ontologies then *complement* the data itself by adding semantics to it. *Schemas*, on the other hand, also provide a descriptive model of the data, but do not restrain to the semantic level: schemas also provide the structure of the data containers in the shape of a grammar [105]. Still, despite their differences, both schemas and ontologies can act as harmonizing mechanisms across research teams dedicated to annotation. This approach is exemplified by the Text Encoding Initiative (TEI) [39] or the Gene Ontology (GO) [51] consortia, that were precisely founded in order to define and maintain domain vocabularies for literary and genetic annotations. Noteworthy, both the TEI and the GO consortia emphasize in their respective documentation that schemas must be steadily *updated*. [127] goes as far as stating that “like databases schemas, ontologies inevitably change over time”. Yet, schemas and ontologies are two different approaches to modelling. On the one hand, *ontologies* are meant to make explicit a certain conceptualization of a domain, by means of a well documented formalism; ontologies then *complement* the data itself by adding semantics to it. *Schemas*, on the other hand, also provide a descriptive model of the data, but do not restrain to the semantic level: schemas also provide the structure of the data containers in the shape of a grammar [105]. While ontology evolution has attracted some attention [169, 127], SE for complex¹ markup data is an untouched field, so far as we know, despite a comparable need in both settings.

Additionally, as illustrated on Figure 8.1, Data Exchange is generally oriented towards the translation of the instances of an original schema into some instance of the new schema, based on a given schema mapping Σ , but not the other way round. Using Σ to derive a backwards transformation from the instances of the new schema to the instances of the older schema has drawn some attention [69, 9, 10], but to the

¹DE has indeed been studied for XML data – see [188, 8].

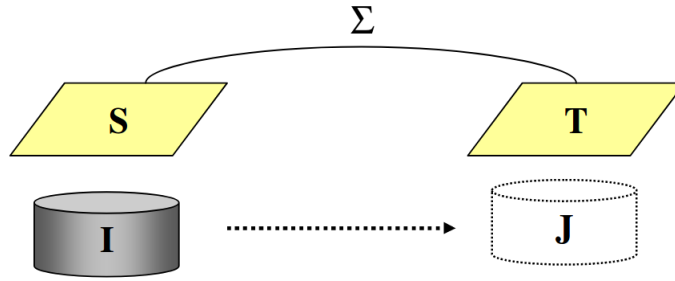


Figure 8.1: The Data Exchange problem. Figure extracted from [108].

best of our knowledge, those studies are focussed on relational databases.

8.2.2 Bidirectional Transformations

As indicated by [57], most bidirectional propositions regarding Data Exchange are query-oriented, in that they consist in establishing “whether an existing transformation specified in a query language, e.g., SQL, Datalog, or XQuery, can be “reversed” in some meaningful way, i.e., create a new transformation from the target of the original transformation to its source, rather than starting with bidirectionality by design”. On the opposite, a wide array of natively bidirectional languages of data transformation have been proposed, with a comparably wide diversity of approaches. The article cited above provides the reader with an interesting panorama of the existing approaches. Native bidirectional transformations are classically defined a mechanism for maintaining the consistency of two (or more) related sources of information [57]. Bidirectional transformations are split into three main families: *Triple graph grammars*, *Algebraic graph transformations* and *Lenses*. Triple graph grammars (TGG) [153] permit to relate two abstract models, that have to be represented as graphs, by means of a third graph that defines the mapping between subgraphs in one abstract model to subgraphs in the second model. The purpose of the correspondence graph is to relate the substructures of the two models that have a one-to-one relation; since model-instance relation is modelled as graph morphism, a substructure from an instance can be related to a substructure from its model if it can be derived from it by means of a series of morphisms; thus, a substructure from an instance can be translated into a substructure from the other domain provided the model-to-model correspondence for the subgraph of the model it relates to. Apart from the ‘positive’ relations between the two models, exclusion rules, that prevent to translate well-formed data from one model into non well-formed data from the other (i.e. that capture the well-formedness rules of each domain and articulate them together) need be defined as well [67]. It results from the above that defining a working TGG is not trivial even in case of two simple and easy to abstract models (class-diagram and relational database models). Additionally, the classic (and actually, mostly unique) example of TGG transformation that is given in the literature offers a means to relate class-diagrams to relational schemas – yet, this very example shows that TGGs are not deterministic (see [153])

p. 416), which is very limiting in terms of synchronization properties. Nonetheless, a synchronization model based on TGGs have been proposed [91], that relies upon deterministic TGGs only.

Lenses [73, 137] originate from the functional language community, as an alternative solution to the Database view-update problem. A Lens can be modelled as a pair of unidirectional transformations. Given a source of information X , the forward transformation or *get*, maps that source structure to a view Y (which can be materialized or abstract), while the backward transformation or *put*, maps a possibly updated view Y back to a source X . Lenses are not bijective in general. In bijective languages, the *put* function shall be injective and the *get* function shall be the corresponding inverse. One strength of bidirectional languages is the fact that the forward transformation can be an arbitrary function. “Since the forward transformation may discard information in general, the backward transformation typically takes two arguments: an updated output as well as the original input” [57]. Hence, classically, *put* and *get* are defined asymmetrically as follows:

$$\begin{aligned} \textit{get} &: X \longrightarrow Y \\ \textit{put} &: Y \times X \longrightarrow X \end{aligned}$$

In the above, the antisymmetry of *put* and *get* in terms of arguments has to be interpreted as follows: Y is a view of X , and as such, it shall always be possible to obtain Y as the output of a function of X . This is confirmed by the laws a lens has to verify, that describe the way a bidirectional transformation shall behave (and hence called *well-behaved* rules):

$$\begin{aligned} \text{PUTGET: } & \textit{get}(\textit{put}(y, x)) = y \\ \text{GETPUT: } & \textit{put}(\textit{get}(x), x) = x \end{aligned}$$

In order to take into account the fact that the view Y can nonetheless be updated, an additional function *create* : $Y \longrightarrow X$ is added, so that:

$$\text{CREATEGET: } \textit{get}(\textit{create}(a)) = a$$

Lenses are designed to work for simple data structures like lists, trees [73, 137] and relational databases [23].

Later on, symmetric lenses were introduced, based on the notion of complement. If the function *get* does not retain all the information in X , then X can somehow be mapped onto two different spaces: Y , by *get*, and a complement space C_X , in which the information that is discarded by *get* is sent. Since we now consider Y of equal importance as X , in this symmetric setting, the same can be said of *put*, that projects Y onto X and C . Thus symmetric lenses are functions

$$\begin{aligned} \textit{get} &: X \times C \longrightarrow Y \times C \\ \textit{put} &: Y \times X \times C \longrightarrow X \times C \end{aligned}$$

so that:

$$\begin{aligned} \text{PUTGET: } & \textit{put}(y, c) = (x, c') \Rightarrow \textit{get}(x, c') = (y, c') \\ \text{GETPUT: } & \textit{get}(x, c) = (y, c') \Rightarrow \textit{put}(y, c') = (x, c') \end{aligned}$$

In other words, a well-behaved symmetric lens shall be so that the absence of update on one domain shall not lead to the propagation of undue modification either on the other domain or on the complement.

It has to be noted that only symmetric bidirectional transformations may fit our use case, in which the instances of the collective schema and of the individually proposed, alternative schemas, are not the views of one another, and may both contain information that cannot be deduced from what is contained in the other domain.

Eventually, *Algebraic graph transformations* have been proposed as a means to tackle graph-structured data; the original proposition emanates from the GroundTram project, under the name of UnQL+ [94]. UnQL+ is based upon the assessment that UnCAL [38], a graph query algebra made out of a series of nine graph constructors together with a conditional and a structural recursion operator, can be bidirectionalized [94]. The authors also added constructors for model transformation (select, replace, extend and delete), also bidirectionalized. The resulting bidirectional language is well-behaved. Yet, the biggest disadvantage of the GroundTram proposition is that UnQL+ is not symmetric. Another limitation of the GroundTram original proposition was its being restricted to unordered graphs. This has lately been compensated, by extending the proposition to ordered graphs as well [187].

8.2.3 Bidirectionalizing eAG/SeAG

We can draw a few conclusive remarks from the above state-of-the-art. First, bidirectionalizing eAG/SeAG, in our application context, demands that the bidirectional transformation we want to derive from the schema transformation shall be symmetric. It shall also work on graphs, obviously – while the combination of the this characteristics and of symmetry has not been studied so far, to the best of our knowledge.

We now turn to the presentation of our proposition, that, contrary to most the above mentioned bidirectional languages, is tailored for one data model. Our proposition, though it is not technically grounded on any other bidirectional transformation language, is philosophically related to symmetric lenses and UnQL+.

From symmetric lenses, it takes the idea that *put* and *get* transformations have to share the same arguments, and that the part of the data each transformation does not translate has to be part of the arguments for the inverse transformation.

From UnQL+, we retain the algebraic and compositional approach: we will define first a set of constructors/operators for SeAG, that can be composed to express complex SeAG amendments (or definitions, but we will not focus on that aspect); then, each operator will be interpreted as an operator on the instances, and bidirectionalized. Since this is an ongoing work, the well-behaved quality of those bidirectional operators will only be suggested by means of some examples, but not proven, unfortunately.

This last chapter will be organised as follows. First, we present the operators for the definition of schema transformations; then, we turn to how to derive a sound eAG transformation out of them, and how to bidirectionalize those eAG transformations.

Chapter 9

SeAG Transformations

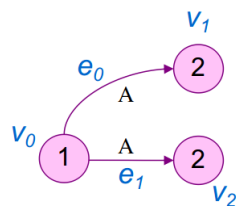
We present here the SeAG transformations that, from a schema S , will enable the definition of a second schema S' . Those transformations will be defined by the composition of elementary operations, called modifications. To each kind of modification corresponds one operator. Those operators work either on rooted, single-leafed sub-graphs of the input schema (*Mod* and *Unite* operator), or on isolate nodes (*Split*). The composition of those modifications, followed by a checking operation aiming at ensuring that the produced graph is a well-formed SeAG, will result in a transformation.

9.1 Matrix-based Representation of eAG/SeAG: Calculability

The upcoming transformation definitions will be expressed using the matrix representation for eAG and SeAG introduced in Paragraph 5.3.0.4. We give a brief summary of that representation here, together with some useful complements.

Finite incidence matrix. Given a graph $G = (V, E)$, so that the set of the node identifiers of G is \mathcal{I}_G and the set of edge identifiers of G is \mathcal{J}_G . The incidence matrix of G on \mathcal{I}_G and \mathcal{J}_G is a matrix $[G]^{\mathcal{I}_G, \mathcal{J}_G}$ whose lines are indexed on the node identifiers of a G and whose columns are indexed on the edge identifiers of the same graph, so that $[G]_{i,j}^{\mathcal{I}_G, \mathcal{J}_G} = 1$ indicates that the node i is the summit of the edge j , and $[G]_{k,j}^{\mathcal{I}_G, \mathcal{J}_G} = -1$ indicates that the node k is the end of that edge.

For example, be G the following graph and its corresponding finite incidence matrix:



$$[G]^{\mathcal{I}_G, \mathcal{J}_G} = \begin{matrix} v_0 & & e_0 & e_1 \\ v_1 & & 1 & 1 \\ v_2 & & -1 & 0 \\ & & 0 & -1 \end{matrix}$$

Infinite matrix. $[G]^{\mathcal{I}_G, \mathcal{J}_G}$ is a $|V| \times |E|$ matrix. Yet, importantly, given a value $x \notin \mathcal{I}_G$, if we define $\mathcal{I}^+ = \mathcal{I}_G \cup \{x\}$, $[G]^{\mathcal{I}^+, \mathcal{J}_G}$ is defined as above for the couples $(i, j) \in \mathcal{I}_G \times \mathcal{J}_G$ and so that $[G]_{i,j}^{\mathcal{I}^+, \mathcal{J}_G} = 0$ if $i = x$. The same applies to \mathcal{J}_G . Recursively, if \mathcal{I} is the set of all possible node and edge identifiers, then the infinite matrix $[G]^{\mathcal{I}, \mathcal{I}}$ is defined.

Finite-infinite matrix equivalence. Conversely, given $[G]^{\mathcal{I}, \mathcal{I}}$, it is possible¹ to obtain $[G]^{\mathcal{I}_G, \mathcal{I}_G}$ by deleting the empty lines and columns of $[G]^{\mathcal{I}, \mathcal{I}}$. One consequence of this property is the possibility to express graph union by means of a classic matrix sum, not on the finite matrices that are not necessarily indexed on the same identifier sets, but on the infinite matrices that, by definition, are. See the item *eAG/SeAG union calculation* below.

Positive and negative restrictions. Be then $[G]$ the (finite or infinite) incidence matrix of a graph G . The positive restriction of $[G]$, denoted $[G^+]$, is defined by $[G^+]_{i,j} \neq 0$ iff $[G]_{i,j} = 1$. Then $[G^+]_{i,j} = 1$. The negative restriction $[G^-]$ is defined symmetrically for negative values.

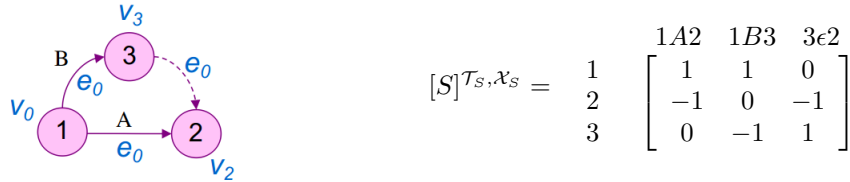
For instance, the negative restriction of the matrix $[G]^{\mathcal{I}_G, \mathcal{J}_G}$ is:

$$[G^-]^{\mathcal{I}_G, \mathcal{J}_G} = \begin{matrix} v_0 \\ v_1 \\ v_2 \end{matrix} \begin{bmatrix} e_0 & e_1 \\ 0 & 0 \\ -1 & 0 \\ 0 & -1 \end{bmatrix}$$

Provided a conventional order on the set of edge identifier, the writing for this matrix can be condensed into: $[G^+]^{\mathcal{I}_G, \mathcal{J}_G} = [-v_1 \ -v_2]$.

Template. Incidence matrices need not be defined on the identifier sets of the nodes and edges of a graph: its lines (resp. columns) just need to be indexed on any set that enable to identify the nodes (resp. edges). In the case of a schema $S = (V_S, E_S)$, for instance, there is a bijection between the set of node identifiers and the set $\mathcal{T}_S = \{type(v), v \in V_S\} \subseteq \mathcal{T}$ of node types of the schema; there is also a bijection between the set of edge identifiers and the set of triples $\mathcal{X}_S = \{(type(sut(e)), label(e), type(end(e))), e \in E_S\} \subseteq \mathcal{T} \times \mathcal{L} \times \mathcal{T}$. Thus, a schema, or any graph for which those bijections exist, can be represented by means of an incidence matrix indexed on those sets.

For instance, the following schema can be described by the corresponding finite incidence matrix:



Interestingly, one can notice that the value 1 in the column relative to the edge characterized by 1A2 above refers to the node whose identifier is 0 and the value -1 refers to

¹Since \mathcal{I} is countable, as well as any set of identifiers \mathcal{I}_G from any graph, we can define an arbitrary partial order $\leq_{\mathcal{I}}$ (resp. $\leq_{\mathcal{I}_G}$) on \mathcal{I} (resp. \mathcal{I}_G), so that $\forall (i, j) \in \mathcal{I} \cap \mathcal{I}_G, i \leq_{\mathcal{I}} j \Leftrightarrow i \leq_{\mathcal{I}_G} j$. Provided that, the translation from the finite to the infinite matrix, and the converse, are deterministic.

the node whose identifier is 2. This can be captured all in one matrix representation, by substituting to 1 and -1 a matrix value, that is the restriction of $[S^+]^{\mathcal{T}_S, \mathcal{X}_S}$ to the edges whose characteristic triple in $\mathcal{T} \times \mathcal{L} \times \mathcal{T}$ matches 1A2 where 1 appears on that column of $[S]^{\mathcal{T}_S, \mathcal{X}_S}$, and similarly, the restriction of $[S^-]^{\mathcal{T}_S, \mathcal{X}_S}$ to the same edges where -1 appears. This gives:

$$\begin{array}{c} 1 \\ 2 \\ 3 \end{array} \begin{array}{ccc} 1A2 & 1B3 & 3\epsilon 2 \\ \left[\begin{array}{ccc} [v_0] & [v_0] & 0 \\ [-v_2] & 0 & [-v_2] \\ 0 & [-v_3] & [v_3] \end{array} \right] \end{array}$$

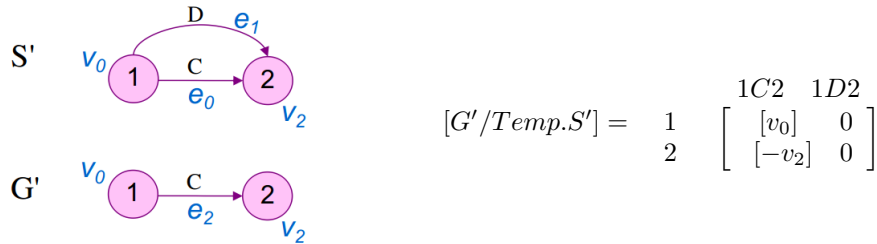
The above representation is called the representation of the schema S on the template of S . The template of S is the finite incidence matrix of S built on $\mathcal{T}_S, \mathcal{X}_S$: it is, in a sense, the outer matrix in that representation. The graph S is represented by replacing the ± 1 values from each column j from that incidence matrix by the positive or negative restriction of the subgraph of S limited to the edges characterized by triple j .

Importantly, given the template of a schema S , any instance of S can be represented on that template. Consider the graph G above². It contains only edges labelled A , joining a node typed 1 to a node typed 2. The representation of G over the template of S , denoted $[G/Temp.S]$, is the following:

$$\begin{array}{c} 1 \\ 2 \\ 3 \end{array} \begin{array}{ccc} 1A2 & 1B3 & 3\epsilon 2 \\ \left[\begin{array}{ccc} [v_0 \ v_0] & 0 & 0 \\ [-v_2 \ -v_1] & 0 & 0 \\ 0 & 0 & 0 \end{array} \right] \end{array}$$

Infinite template. Be a graph G and a schema S . Just as for incidence matrices, it is possible to index the template of a schema not on the sets \mathcal{T}_S and \mathcal{X}_S representing the type and label values that can be found in S , but on the set of all possible types and labels \mathcal{T} and \mathcal{L} ; the inner matrices, that correspond to the positive or negative restrictions of the incidence matrices of some subgraph of G , can be translated into their infinite form, as defined above. Hence the notion of infinite template representation of a graph. Let us denote the representation of G' over the infinite template of S' : $[G'/Temp.S']^\infty$.

eAG/SeAG union calculation. Consider the following schema-instance pair G', S' , and its corresponding finite template-based representation:



²... That is not a well-formed eAG, but let us do as if it were, for the sake of simplicity...

Since the template lines and columns are not indexed on the same sets as the lines and columns of the representation of G over the template of S provided above, the sum of the two block matrices is not defined and would be meaningless. On the contrary, it is possible to express G' over the infinite template of S' and G over the infinite template of S : both representations are then infinite block matrices, whose blocks are infinite matrices, as follows:

$$[G'/Temp.S']^\infty = \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ \vdots \end{array} \begin{array}{c} \dots \quad 1A2 \quad 1B3 \quad 3\epsilon 2 \quad 1C2 \quad 1D2 \quad \dots \\ \left[\begin{array}{cccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & [v_0] & 0 \\ 0 & 0 & 0 & 0 & 0 & [-v_2] & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right] \end{array}$$

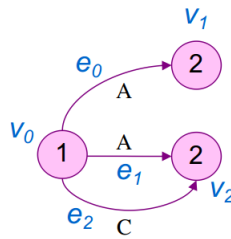
Similarly, we also have:

$$[G/Temp.S]^\infty = \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ \vdots \end{array} \begin{array}{c} \dots \quad 1A2 \quad 1B3 \quad 3\epsilon 2 \quad 1C2 \quad 1D2 \quad \dots \\ \left[\begin{array}{cccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ [v_0 \ v_0] & 0 & 0 & 0 & 0 & 0 & 0 \\ [-v_2 \ -v_1] & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right] \end{array}$$

The, it is possible to define a sum of the two, where the '+' operator is a term-to-term application of the operator defined in table 9.1:

$$[G'/Temp.S']^\infty + [G/Temp.S]^\infty = \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ \vdots \end{array} \begin{array}{c} \dots \quad 1A2 \quad 1B3 \quad 3\epsilon 2 \quad 1C2 \quad 1D2 \quad \dots \\ \left[\begin{array}{cccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ [v_0 \ v_0] & 0 & 0 & 0 & 0 & [v_0] & 0 \\ [-v_2 \ -v_1] & 0 & 0 & 0 & 0 & [-v_2] & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right] \end{array}$$

which represents the following graph:



Noteworthy, this graph is the union of the graphs G and G' (that is validated by the union of S and S' , which could be calculated the same way, by summing the template-based representations of S and S').

+	-1	0	1
-1	-1	-1	0
0	-1	0	1
1	0	1	1

Table 9.1: The term-to-term addition operated for summing two incidence matrices. This operator ensures to keep the result of any sum of values from the set $\{-1, 0, 1\}$, which is the set of values of the elements of an incidence matrix, in the same set.

Based on those considerations, that may somehow give some clues about how schema and instance will be manipulated, we can turn to the definition of SeAG transformations first.

9.2 Composing Modifications: General Strategy

The definition of a transformation can thus be modelled as follows:

$$S \xrightarrow{\text{modification}_1} G_1 \xrightarrow{\text{modification}_2} G_2 \dots \xrightarrow{\text{modification}_N} G_N \xrightarrow{\text{check}} S'$$

It is natural that a modification shall not be required to produce (at each step of this sequence of modifications) a well-formed schema. Yet it is also natural that all the G_i graphs above have to be modifiable, in the following sense: it has to belong to the definition space of any other operator. This implies: 1) that operators shall be defined on a wider set of (modifiable) graphs than schemas and 2) they shall produce graphs belonging to the same set of modifiable graphs.

We go with the following definition of what will have to be modifiable graphs for schema operators, that we call *casts*.

Definition 9.1: casts. A cast is any graph that that is expressible on its own template.

Property 9.1. Be $G = (V, E)$ a graph so that $A \neq \emptyset$. We have the following properties:

1. If there is a schema S so that $G \subseteq S$, then G is a cast.
2. G is a cast iff $\exists \phi : \mathcal{T} \times \mathcal{L} \times \mathcal{T} \longrightarrow E$ so that ϕ is injective.

Comment. The class of *cast* graphs is a good candidate for modifiable graphs for the SeAG operators, since it corresponds to the widest class of graphs that can be handled in a homogeneous ways with schemas, that is, by means of their template. The general idea is that since transformations on schemas will have to be interpreted as modifications on the instances of the schema, they can be defined as modifications of the template of the schema: the instances of a schema being expressible on the

template of that schema, they will in turn be impacted correspondingly – in first approximation.

9.3 *Mod* Operator

The first kind of operation one may want to apply on a schema, in order to modify it, is to substitute a subgraph of that schema by a novel one. We propose here a definition of such a substitution operator, called *Mod*, that takes two arguments: the subgraph to be ‘retrieved’ and the subgraph to be inserted instead. We first define the subgraphs that can be used as arguments for the *Mod* operator, that we call schematic cells.

9.3.1 Schematic Cells

Be the following preliminary definition:

Definition 9.2 : Notion of walk. A **walk** on a connected, rooted, single-leafed graph $G = (V, E)$ is any sequence \wp built on the set E , so that:

- $sut(\wp_1) = root(G)$
- $end(\wp_{|\wp|}) = leaf(G)$
- $\forall n \in [1; |\wp|], end(\wp_{n-1}) = sut(\wp_n)$

Notation. Be a cast G . Be v, v' two nodes of G so that there is a subgraph G' of G rooted in v and possessing v' as its only leaf, and so that there exists a walk \wp on G' . Synthetically, we will then say that there is a walk on G between the nodes v and v' , denoted $\exists \wp \stackrel{prom.}{\subset} G$.

A node v will be said to belong to a walk \wp iff $\exists e \in \wp$ so that $v = sut(e) \wedge v = end(e)$. The fact v belongs to \wp will be denoted $v \in \wp$.

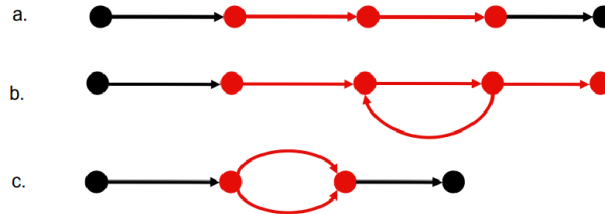
The *Mod* operator is designed as a substitution operator, working on a particular set of subgraphs of a schema. Those subgraphs, called schematic cells, are defined as follos:

Definition 9.3 : Schematic cell. Be a cast G . A *schematic cell* (or *cell*) of G is a subgraph $\chi \subseteq G$ so that:

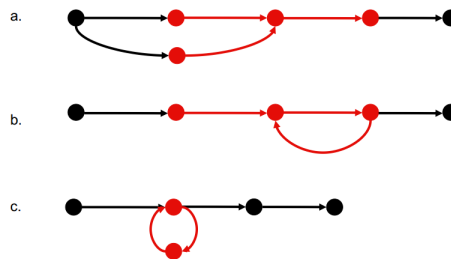
- χ is empty
- OR χ is so that:
 1. χ is rooted and single-leafed;
 2. χ is connected;
 3. χ is not restricted to one node;
 4. $\forall e_0$ edge χ , if $\exists \wp, \wp'$ two walks so that $e_0 \in \wp \wedge e_0 \in \wp'$, then $\exists \wp'' \mid \{e \in \wp\} \subseteq \{e \in \wp''\}$ et $\{e \in \wp'\} \subseteq \{e \in \wp''\}$.

Property 9.2 Be a cast G and χ a cell included in G . Then χ is also a cast.
Proof. This is true of any subgraph of a cast.

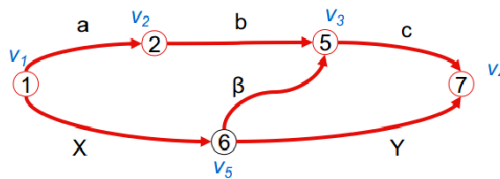
Examples. We provide a few examples of schematic cells (in red). Noteworthily, those may contain cycles.



Counter examples. The following graphs are not cells because: a. they are not rooted; b. there is not leaf; c. idem.



The following case is also excluded, since some of its edges belong to two different walks that do not differ by the fact one contains a cycle the other does not:



The definition of a cell, even if it imposes cells to abide by a strict graph model, does not refrain cells to have strong connexions with the rest of the schema. In order to draw a typology of cells regarding the way they are connected to the rest of the schema, and to study how to deal with each kind of cell accordingly, we split the notion of cells into two sub-categories: independent and non-independent cells.

Definition 9.4 : Independant cells. Be $G = (V, E)$ a cast and et $\chi = (V' \subseteq V, E' \subseteq E)$ a cell of G . The cell χ is independant iff:

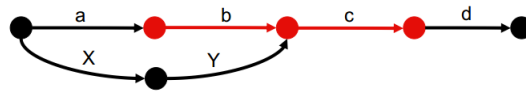
1. $\forall e \in E \setminus E' \mid \exists v \in V, v' \in V', v[e]v' \in S$, then $v' \in \{\text{root}(\chi), \text{leaf}(\chi)\}$;
2. $\forall e \in E \setminus E' \mid \exists v \in V, v' \in V', v'[e]v \in S$, then $v' \in \{\text{root}(\chi), \text{leaf}(\chi)\}$.

Comment. To enlighten the definition of independent cells, let us focus on a simple case of such cells, that is, an independent, linear cell (containing no cycle): Be S a schema. Be χ a linear independent cell and be e_1 et e_2 two edges of χ . Be $G = (V_G, E_G) \mid G \subseteq_{\text{leaf}(S)}^{\text{root}(S)} S$ a connected rooted and single-leafed subgraph of S . Then:

$$e_1 \in E_G \Leftrightarrow e_2 \in E_G$$

In other words, if we reason in terms of the annotation language of the schema, the sequence of labels obtained by reading the edges of the cell can be regarded as one letter of the alphabet, in that the sequence either belongs as a whole or not at all to the words of the annotation language of S . In this particular case, modifying the cell as a whole thus makes sense. This applies not only to linear independent cells, but to all independent cells: in the general case, an independent cell represents not only sequences that can be modified as a bulk, but small regular expressions that can be isolated and modified as a bulk.

On the contrary, non-independent cells will be characterized by the existence of edges of S that do not belong to the cell but that point towards nodes that are different from the root and leaf of the cell. Here is an example:



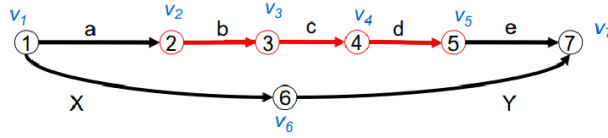
Here, we can sense that the above discussion on how the edges of an independent cell only participate together to the language of the schema does not apply to non-independent cells: the first edge of the cell, labelled **b**, belongs to only one word of the annotation language of S , while the second edge, labelled **c**, belongs to two words. This question will be studied in detail in Paragraph 9.3.3.

9.3.2 *Mod* Operator: Intuitive Presentation

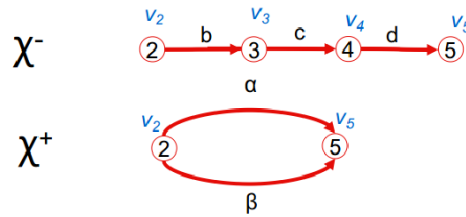
As stated above, *Mod* is a substitution operator. It basically works by extracting a cell from a schema and by inserting another cell, sharing the same root and leaf, instead. Since cells can be empty graphs, *Mod* covers three different semantic modifications on schemas, depending on the arguments given as input to the operator: deletion, insertion and substitution properly speaking.

We present the general principle of the way the *Mod* operator works, based on the simplest case, that is, when applied to an independent schematic cell.

As a basis for illustration, consider the following schema S . Let us consider that the editor wants to replace the pattern “**b** followed by **c** followed by **d**” by a pattern expressing “ $\alpha|\beta$ ”.



This transformation, that actually corresponds to a *Mod* operation, can be defined as follows: it consists in substituting to a cell χ^- constituted of the edges expressing the sequence “bcd” in S a cell χ^+ describing the regular expression $\alpha|\beta$. The two cells, with the corresponding node types, are represented below.



The templates of the three graphs to be considered here, namely $S = (V_S, E_S)$, $\chi^- = (V_-, E_-)$ and $\chi^+ = (V_+, E_+)$, are the following:

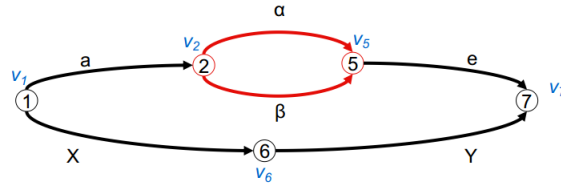
$$\begin{aligned}
 [S/[Temp.S]] &= \begin{matrix} & a & b & c & d & e & X & Y \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix} & \left[\begin{array}{cccccc} [v_1] & [0] & [0] & [0] & [0] & [v_1] & [0] \\ -[v_2] & [v_2] & [0] & [0] & [0] & [0] & [0] \\ [0] & -[v_3] & [v_3] & [0] & [0] & [0] & [0] \\ [0] & [0] & -[v_4] & [v_4] & [0] & [0] & [0] \\ [0] & [0] & [0] & -[v_5] & [v_5] & [0] & [0] \\ [0] & [0] & [0] & [0] & [0] & -[v_6] & [v_6] \\ [0] & [0] & [0] & [0] & [0] & [0] & -[v_7] \end{array} \right] \end{matrix} \\
 [\chi^-/[Temp.\chi^-]] &= \begin{matrix} & b & c & d \\ \begin{matrix} 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \left[\begin{array}{ccc} [v_2] & [0] & [0] \\ -[v_3] & [v_3] & [0] \\ [0] & -[v_4] & [v_4] \\ [0] & [0] & -[v_5] \end{array} \right] \end{matrix} \\
 [\chi^+[Temp.\chi^+]] &= \begin{matrix} & \alpha & \beta \\ \begin{matrix} 2 \\ 5 \end{matrix} & \left[\begin{array}{cc} [v_2] & [v_2] \\ -[v_5] & [v_5] \end{array} \right] \end{matrix}
 \end{aligned}$$

All those representations can then be translated into representations on infinite templates, as defined in Paragraph 9.1. Then, the following operation implements the fact

that during the transformation, χ^- is deleted in S while χ^+ is inserted:

$$\begin{aligned}
 [M] &= [S/[Temp.S]]^\infty - [\chi^-/[Temp.\chi^-]]^\infty + [\chi^+/[Temp.\chi^+]]^\infty \\
 &= \begin{matrix} & a & b & c & d & e & X & Y \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix} & \begin{bmatrix} [v_1] \\ -[v_2] \\ [0] \\ [0] \\ [0] \\ [0] \\ [0] \end{bmatrix} & \begin{bmatrix} [0] \\ [0] \\ [0] \\ [0] \\ [0] \\ [0] \\ [0] \end{bmatrix} & \begin{bmatrix} [0] \\ [0] \\ [0] \\ [0] \\ [0] \\ [0] \\ [0] \end{bmatrix} & \begin{bmatrix} [0] \\ [0] \\ [0] \\ [0] \\ [0] \\ [0] \\ [0] \end{bmatrix} & \begin{bmatrix} [0] \\ [0] \\ [0] \\ [0] \\ [v_5] \\ [0] \\ -[v_7] \end{bmatrix} & \begin{bmatrix} [v_1] \\ [0] \\ [0] \\ [0] \\ [0] \\ -[v_6] \\ [0] \end{bmatrix} & \begin{bmatrix} [0] \\ [0] \\ [0] \\ [0] \\ [0] \\ [v_6] \\ -[v_7] \end{bmatrix} \end{matrix}^\infty + \begin{matrix} & \alpha & \beta \\ \begin{matrix} 2 \\ 5 \end{matrix} & \begin{bmatrix} [v_2] \\ -[v_5] \end{bmatrix} & \begin{bmatrix} [v_2] \\ -[v_5] \end{bmatrix} \end{matrix}^\infty \\
 &\equiv \begin{matrix} & a & \alpha & \beta & e & X & Y \\ \begin{matrix} 1 \\ 2 \\ 5 \\ 6 \\ 7 \end{matrix} & \begin{bmatrix} [v_1] \\ -[v_2] \\ [0] \\ [0] \\ [0] \end{bmatrix} & \begin{bmatrix} [0] \\ [v_2] \\ -[v_5] \\ [0] \\ [0] \end{bmatrix} & \begin{bmatrix} [0] \\ [v_2] \\ -[v_5] \\ [0] \\ [0] \end{bmatrix} & \begin{bmatrix} [0] \\ [0] \\ [v_5] \\ [0] \\ -[v_7] \end{bmatrix} & \begin{bmatrix} [v_1] \\ [0] \\ [0] \\ -[v_6] \\ [0] \end{bmatrix} & \begin{bmatrix} [0] \\ [0] \\ [0] \\ [v_6] \\ -[v_7] \end{bmatrix} \end{matrix}
 \end{aligned}$$

This represents the following graph expressed on its own template, which does correspond to the initial schema where the sequence bcd is replaced by $\alpha|\beta$.



Thanks to this example, we have introduced the way *Mod* modifications on schemas can be performed. We will now study formally the *Mod* operator in the general case, that is, when it is not required that the cells shall be independent.

9.3.3 *Mod*: Formal Definition in the General Case

Let us first highlight the problem raised by non-independent cells, for the definition of *Mod*.

9.3.3.1 Non Independent Schematic Cells: the Problem they Raise

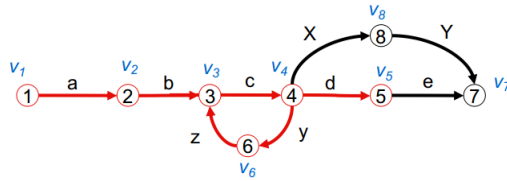
In the previous paragraph, we have sketched how an elementary *Mod* operation should work: given a schema S , such an operation consists in isolating a subgraph χ^- called a cell of S , deleting it and inserting, between the nodes that used to be its root and leaf, another graph χ^+ , that only shares its root and leaf with the pre-existing schema. Now χ^- can be either independent or dependent. In a few words, it is independent if substituting the whole subgraph χ^- of S by another cell only affects the words from the annotation language that imply the execution of at least one root-to-leaf path of the cell. To enlighten this remark, one may consider the non-independent cell provided

page 226: replacing χ^- , in red, by a single edge Ω would imply to lose the annotation word “ $XYcd$ ” defined by the initial schema – word that does not imply the execution of any of the root-to-leaf paths of χ^- .

Semantics of a transformation. As previously shown, a SeAG can be seen as an automaton defining an annotation language (that is, authorized sequences of labels that will then be found in the instances). As a consequence, a transformation, or a modification, can be interpreted as the sign of the will to modify the annotation language of a schema. More precisely, a modification shall ideally enable to target precisely the part of the language that shall be impacted, that is, the words requiring the execution of a root-to-leaf path of the schematic cell to be remove or replaced, and only those words. Impacting other words shall be considered, from this point of view, a side effect of the modification.

Moreover, a schema transformation is intended to be propagated to the instances: for any sequence of labels, in a schema S , that is altered by a modification, all the corresponding label sequences in an annotation will be altered a similar way. In other words, if there are words of the schema language that do not result from the execution of a root-to-leaf path of the original schematic cell, and that are altered as a side effect of a modification operated on the schema, then all the instances of these words will also, as the propagation of this side effect, be altered in the annotation graphs. hence the need to be able to limit, and to quantify, the collateral loss in the schema vocabulary resulting from the definition of a given modification, since this abstract loss will be translated in the loss of editorial information in the instances.

Example. Let us consider the following schema and cell (in red).



Let us then consider a substitution cell χ^+ , made out of a root (typed 1), a leaf (typed 4) and an edge labelled α . Two consequences can be drawn from the above comment:

1. If S' is the schema obtained by substituting χ^- by χ^+ , then the vocabulary of S' shall not contain the word $abcde$: hence, the edge sequence $abcd$ will have to not be present in S' .
2. Instead, the word ae will have to be part of the language of S' .

Operating the strategy of substitution sketched in Paragraph 9.3.2 is not convincing in this case (where χ^- is not independent): the expression $[S/[Temp.S]^{\mathcal{T},\mathcal{L}}] - [\chi^-/[Temp.\chi^-]^{\mathcal{T},\mathcal{L}}] + [\chi^+/[Temp.\chi^+]^{\mathcal{T},\mathcal{L}}]$ will define a graph in which the sequences $abc(yz)^*XY$ are deleted, which, in the perspective of propagating this modification to the instances of the schema, implies that the annotations containing those sequences will be lost.

The target is then to define the image of a given schema by a modification Mod is those to define the cast in which the label sequences that have to be eliminated are indeed missing, with minimal side effect on the annotation words that do not imply the execution of a root-to-leaf path of χ^- .

9.3.3.2 Connectivity Factor

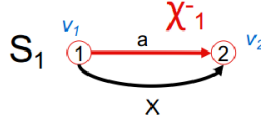
The fact that substituting a non-independent cell may have side effects is due to the fact that there are exogeneous edges – that is, edges of S that do not belong to the cell – that point towards its nodes or, in other words, that some of the edges of a non-independent cell belong to paths that do not go through either the root or the leaf of the cell.

This suggests that such edges of the cell should not, if it is possible, should be preserved after the substitution is performed. We study how to do so hereafter. In the whole paragraph, we consider a cast $G = (V, E)$ and a schematic cell $\chi^- = (V_-, E_-) \subseteq G = (V_G, E_G)$.

Definition 9.4 : Exogeneous edges. We define the two following subsets of $E \setminus E^-$:

1. Out-exogenous edges relatively to χ^- : $\{e \in E_G \setminus E_- \mid \text{end}(e) \in V_-\}$;
2. In-exogenous edges relatively to χ^- : $\{e \in E_G \setminus E_- \mid \text{sut}(e) \in V_-\}$

Example. Some edges of G may belong to both sets: consider the edge labelled X below:



Property 9.3. Be $[G/[Temp.G]]^\infty$ et $[\chi^-/[Temp.\chi^-]]^\infty$ the representations of G and χ^- in their own template. Be $T(\chi^-) = \{t \in \mathcal{T} \mid \exists v \in V_-; \text{type}(v) = t\}$. Then, the exogenous edge sets are characterized as follows:

1. In-exogenous edges : those are the edges $e \in V_G$, associated with the j^{th} triple (t, l, t') in $\mathcal{T} \times \mathcal{L} \times \mathcal{T}$, so that:
 - $t' \in T(\chi^-)$
 - $\exists i, [[G/[Temp.G]]_{i,j}^\infty] < [0]$
 - $\forall i, [[\chi^-/[Temp.\chi^-]]_{i,j}^\infty] = [0]$
2. Out-exogenous edges : those are the edges $e \in V_G$, associated with the j^{th} triple (t, l, t') in $\mathcal{T} \times \mathcal{L} \times \mathcal{T}$, so that:
 - $t \in T(\chi^-)$
 - $\exists i, [[G/[Temp.G]]_{i,j}^\infty] > [0]$
 - $\forall i, [[\chi^-/[Temp.\chi^-]]_{i,j}^\infty] = [0]$

Example. Let us consider the representation of the schema S_1 defined in the previous example, over its own template. In red, we highlight the sub-matrices that represent the subgraph χ_1^- . Based on the above characterization, the edge identified by the triple $(1, X, 2)$ is both an in- and out-exogenous edge relatively to χ_1^- :

$$\begin{array}{l} 1 \\ 2 \end{array} \begin{array}{cc} a & X \\ \left[\begin{array}{cc} [v_1] & [v_1] \\ -[v_2] & -[v_2] \end{array} \right] \end{array} \quad \begin{array}{l} \leftarrow \text{l'arc labellisé } X \text{ est exogène issu de } \chi^- \\ \leftarrow \text{l'arc labellisé } X \text{ est exogène pointant vers } \chi^- \end{array}$$

Property 9.4. A schematic cell χ^- is independent iff the set of exogenous edges relatively to χ^- is included in $\{root(\chi^-), leaf(\chi^-)\}$.

Example. χ_1^- is independent, in the previous example.

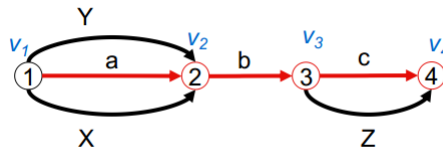
Definition 9.5 : In- and Out- exogenous node degrees. Be G a cast and χ a cell of G . Be v a node from χ . The **in-exogenous degree** of v is the number of in-exogenous edges of G whose end is v . Symmetrically, the **out-exogenous degree** of v is the number of out-exogenous edges of G whose summit is v . Let us denote those degrees $int_{\chi^- \subseteq G}^{exog.}(v)$ et $ext_{\chi^- \subseteq S}^{exog.}(v)$ respectively.

Calcul. Avec les notations précédentes, et en notant i l'indice de $type(v)$ dans \mathcal{T} :

1. $ext_{\chi^- \subseteq G}^{exog.}(v) = |\{[(G/[Temps.G]) - [\chi^- [Temp.\chi^-]])_{i,k} > [0]; k \in \mathbb{N}\}|$
2. $int_{\chi^- \subseteq G}^{exog.}(v) = |\{[(G/[Temps.G]) - [\chi^- [Temp.\chi^-]])_{i,k} < [0]; k \in \mathbb{N}\}|$

Based on those definitions, we can now define the notion of connectivity degree of the nodes of a schematic cell. As indicated above, the edges of a cell implied in some root-to-leaf path of the cast they belong to, that do not go through both the root and leaf of the cell, shall be identified, in order to be, as much as possible, preserved after the substitution operation. The notion of connectivity degree precisely aims at identifying those edges, but also to give a quantitative indication of the amount of words of the annotation language of the schema that do not imply the execution of a root-to-leaf path of the cell, that require the execution of those edges. The idea being: the more words they belong to, the higher the necessity to be preserved, in order to minimize the side-effects of a substitution.

To make this point clearer, let us consider the following example:



In this schema S , the schematic cell χ^- is a single path, possessing the same root and the same leaf as S : thus, the only word resulting from the execution of (a root-to-leaf path of) χ^- is **abc**.

Besides, χ^- is not an independent cell:

- the edge labelled **a** is involved in two words of the annotation language of the schema: **abc** and **abZ**;
- the edge labelled **b** is involved in six words: **abc**, **abZ**, **Xbc**, **XbZ**, **Ybc** and **YbZ**;
- the edge labelled **c** is involved in three words: **abc**, **Xbc** and **Ybc**.

Thus, shall we rank the edges of χ^- by the number of words belonging to the language of S but not of χ^- , that shall be affected by the deletion of the edge in question, we would say that b is worst and c is the best.

However, counting the exact number of words (with special consideration for words resulting from a cycle), in the language of the whole cast, to which each edge of a cell is involved in, does not scale for bigger schemas.

The notion of connectivity degree precisely aims at bypassing this obstacle, by providing a local approximation of the amount of words the deletion of each edge of a cell may impact – as we will see below.

Definition 9.6 : follow and precede sets. Be a connected, rooted and single-leafed graph $C = (V, E)$. For each $v \in V$, let us define two sets $follow_C(v)$ and $precede_C(v)$ by:

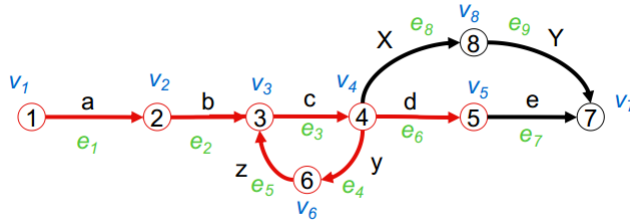
- $follow_C(v) = \{v' \mid \exists \varphi \stackrel{prom.}{\subset}_{leaf(C)}^v C \text{ so that } v' \in \varphi\}$.
- $precede_C(v) = \{v' \mid \exists \varphi \stackrel{prom.}{\subset}_v^{root(C)} C \text{ so that } v' \in \varphi\}$.

Definition 9.7 : Connectivity factor of an edge of a schematic cell.

Be G a cast and χ^- a schematic cell of G . For each edge e de χ^- , the connectivity factor of e , denoted $c_{\chi^-}(e)$, is the value in \mathbb{C} defined as follows³

$$c_{\chi^-}(e) = \sum_{v \in follow_{\chi^-}(end(e))} ext_{\chi^-}^{exog.}(v) + \sqrt{-1} \cdot \sum_{v \in precede_{\chi^-}(sut(e))} int_{\chi^-}^{exog.}(v)$$

Example. Let us consider the following schema an non-independent cell:



³Though improper, we will use the symbol $\sqrt{-1}$ to denote the imaginary unit – the usual notation for it, i , might indeed be confusing in our context, where i is also a usual index for matrix lines...

Only v_4 and v_5 possess an exogenous degree different from zero: then, if the leaf of an edge belongs to a path leading to one of those nodes, then the connectivity factor of those edges is incremented by 1 (2 if the leaf of the edge leads to both). Yet if the leaves of $e_1 \dots e_5$ precede both nodes, the leaf of e_6 only precedes v_5 . Thus, the value of the connectivity factor of $e_1 \dots e_5$ will equal 2 while e_6 will have 1 only.

Definition 9.8: partially independent schematic cells. Be $G = (V_G, E_G)$ a cast and $\chi = (V, E)$ a schematic cell of G . χ^- is said to be **partially independent** iff $\exists E' \subseteq E$ so that:

$$\begin{aligned} & \forall e \in E, e' \in E', \text{ so that } e' \in \text{follow}_\chi(e) \vee e \in \text{follow}_\chi(e'), \\ & \exists (n, m) \in \mathbb{N}^2 \mid c_\chi(e) - c_\chi(e') = n + \sqrt{-1} \cdot m \vee c_\chi(e') - c_\chi(e) = n + i \cdot m \end{aligned}$$

Nota. In case of an independent cell, $\forall e \in E, e' \in E', \text{ so that } e' \in \text{follow}_\chi(e) \vee e \in \text{follow}_\chi(e'), \exists (n, m) \in \mathbb{N}^2 \mid c_\chi(e) - c_\chi(e') = 0$.

Interpretation. The factor $c_\chi(e)$ enables to identify some contextual characteristics of e :

1. If we consider e alone:
 - $Re(c_\chi(e))$ is the number of out-exogeneous edges relatively to χ , originating in nodes of χ that the leaf of e precedes. Any such bifurcation, if it is located ‘after’ the leaf of the edge e but before the leaf of χ , implies that e belongs to at least one word of the annotation language that does not result from the execution of a root-to-leaf path of χ .
 - Symmetrically, $Im(c_\chi(e))$ is the number of in-exogeneous edges relatively to χ , originating in nodes of χ that precede the summit of e . Any such bifurcation, if it is located ‘before’ the summit of the edge e but after the root of χ , implies that e belongs to at least one word of the annotation language that does not result from the execution of a root-to-leaf path of χ .
2. If we consider the whole set of edges of χ : be e, e' two edges of χ , so that, for instance, $sut(e') \in \text{follow}_\chi(\text{leaf}(e))$. Then:

$$\begin{aligned} Re(c_\chi(e)) = Re(c_\chi(e')) & \Leftrightarrow \sum_{v \in \text{follow}_\chi(\text{end}(e))} ext_\chi^{exog.}(v) = \sum_{v \in \text{follow}_\chi(\text{end}(e'))} ext_\chi^{exog.}(v) \\ \text{Yet } sut(e') \in \text{follow}_\chi(\text{end}(e)) & \Leftrightarrow \text{follow}_\chi(\text{end}(e')) \subseteq \text{follow}_\chi(\text{end}(e)) \\ \text{Hence } Re(c_\chi(e)) = Re(c_\chi(e')) & \Leftrightarrow \sum_{v \in \text{follow}_\chi(\text{end}(e)) \setminus \text{follow}_\chi(\text{end}(e'))} ext_\chi^{exog.}(v) = 0 \end{aligned}$$

This is verified in two cases:

- (a) e and e' belong to a cycle included in χ (in this case, $\text{follow}_\chi(\text{end}(e)) = \text{follow}_\chi(\text{end}(e'))$, by definition of *follow*).

(b) e and e' do not belong to a cycle of χ and all the nodes that belong to $follow_\chi(end(e)) \setminus follow_\chi(end(e'))$ have an out-exogenous degree equal to zero – in other words, all the exogenous edges going out of χ bifurcate before both e and e' , or on a path of χ to which none of e and e' belong.

Symmetrically, $Im(c_\chi(e)) = Im(c_\chi(e'))$ is verified in only two cases:

(a') e and e' belong to a cycle included in χ .

(b') e et e' do not belong to a cycle of χ and all the nodes that belong to $precede_\chi(sut(e')) \setminus precede_\chi(sut(e))$ have an in-exogenous degree equal to zero – in other words, all the exogenous edges pointing towards χ do so after both e and e' , or on a path of χ to which none of e and e' belong.

Then, let us consider the case where $e' \in follow_\chi(end(e))$, $c_\chi(e) = c_\chi(e')$, so that e and e' do not belong to a cycle of χ . **The fact that the connectivity factor of e and e' is equal means that:**

$$\forall G \subseteq_{leaf(S)}^{root(S)}, \quad e \in E_G \Leftrightarrow e' \in E_G$$

Thus, and that is the main conclusion of this interpretation paragraph: if two edges $e, e' \in \chi$ are so that $e' \in follow_\chi(e) \vee e \in follow_\chi(e')$, and:

$$Im(c_\chi(e)) \leq Im(c_\chi(e')) \text{ AND } Re(c_\chi(e)) = Re(c_\chi(e'))$$

that is so that $\exists(n, m) \in \mathbb{N}^2 \mid c_\chi(e) - c_\chi(e') = n + i.m$,

then it means that any path of the cast e belongs to also contains e' , or that all path containing an edge belonging to a cycle e belongs to also contains an edge of a cycle e' belongs to (etc.).

On the contrary, given two edges e and e' so that $Re(c_\chi(e)) > Re(c_\chi(e'))$, then it means that there is a node between the end of e and the summit of e' (included) that is the summit of an out-exogenous edge; this means, in particular, that the edge e is involved in a word of the annotation language of the whole graph that does not involve, in particular, the execution of the edge e' . And symmetrically if $Re(c_\chi(e)) > Re(c_\chi(e'))$.

Thus, the existence of a set E' defined in Definition 9.8 is highly interesting for performing a modification that shall prevent from the execution of the root-to-leaf paths of the original cell, without discarding words that do not result from the execution of such a path. Indeed, the existence of E' means that there is a set of edges that come before the out-exogenous edges all the other edges of the cell come before, and only those (that is, the out-exogenous edges that originate in the leaf of the cell), and that come after the in-exogenous edges all the the edges of the cell come after (that is, the ones that end on the root of the cell); in other words, E' contains the edges of the cell that are involved in all and only in the root-to-leaf paths of the cell. Deleting those edges will thus prevent from keeping words resulting from the execution of the cell, while maintaining all the other edges of the cell (that participate in defining exogenous words) – which is exactly the behaviour we expect from the *Mod* operator.

Based on those considerations, let us now study how *Mod* shall be defined for such schematic cells, before dealing with the case of cells that are neither independent, not partially independent.

Definition 9.9 : Ambiguous cells. A cell that is not partially independent (and thus, not independent either) is called an ambiguous cell.

9.3.3.3 *Mod* for Partially Independent Schematic Cells

Mod is designed as a substitution operator: it is applied on a cast G , and takes two arguments, an original cell χ^- and a replacement cell χ^+ . In the case where χ^- is independent, as illustrated in Paragraph 9.3.2, *Mod* works by deleting χ^- from the G and by inserting, between the nodes that were the root and the leaf of χ^- , the replacement cell χ^+ .

We adapt this procedure to a partially independent cells hereafter.

Property 9.5. be G a cast and χ a cell of G . $\forall \chi, \exists I \in \mathcal{P}(\mathbb{N}), \exists \{G_i = (V_i, E_i)\}_{i \in I}$ so that:

1. $\forall i \in I, \text{root}(G_i) = \text{root}(\chi) \wedge \text{leaf}(G_i) = \text{leaf}(\chi)$;
2. $\bigcup_{i \in I} G_i = \chi$;
3. $\forall i \neq j, E_i \cap E_j = \emptyset$;
4. $\forall i \in I, \exists \varphi \stackrel{\text{prom.}}{\subseteq}_{\text{leaf}(\chi)}^{\text{root}(\chi)} G_i$ so that $\{e \in \varphi\} = E_i$.

Proof. It is always possible to decompose a rooted, single-leafed and connected graph into a set of paths. The peculiar aspect of the above decomposition is there is no pair of paths that share an edge. This property can be verified because of the property number 3. in the definition of schematic cells (see p. 224).

Property 9.6. For a given schematic cell, the above decomposition is unique.

Preuve. The decomposition can be obtained as follows: find the biggest connected subgraphs of the cell whose leaf is the leaf of the cell, and rooted in each of the ends of the edges of the cell whose summit is the root of the cell. This set of biggest connected subgraph is unique.

Property 9.7. B G a cast, χ a cell and $\{G_i\}_{i \in I}$ the decomposition of the cell into linear, non-intersecting subgraphs. Then χ is ambiguous iff $\exists j \in I$ so that G_j is ambiguous.

Preuve. If for any $G_i = (V_i, E_i)$, G_i is partially independent, that is if for all i : $\exists E'_i \subseteq E_i$ so that:

$$\forall e \in E_i, e' \in E'_i, \text{tels que } e' \in \text{follow}_\chi(e) \vee e \in \text{follow}_\chi(e'), \quad \exists (n, m) \in \mathbb{N}^2 \mid c_\chi(e) - c_\chi(e') = n + \sqrt{-1} \cdot m \vee c_\chi(e') - c_\chi(e) = n + \sqrt{-1} \cdot m,$$

then, since two graphs G_i and G_k of the decomposition have a void intersection, then the above property holds for the union of G_i and G_k , and thus, for χ that is also partially independent.

Important precision. If the decomposition of a cell contains more than one graph, then, Mod will be applied separately on each of those graphs – which enables to define Mod for cells whose decomposition contains one single graph. One may check, in the following calculations, that this will not lead to the multiple insertion of the substitution cell χ^+ , since Mod will make use of the addition operator defined in Paragraph 9.1 for the insertion of χ^+ , that works as the union operator for graphs: inserting the same graph by means of this addition, when performing the substitution of each subgraph of a cell, will have the same result as performing the union of a graph with itself, that is, none.

Property 9.8 Be $G = (V_G, E_G)$ a cast and $\chi = (V, E)$ a partially independent schematic cell of G whose decomposition into non-intersecting linear graphs contains one graph (the cell itself). Then there is a maximal set $\exists E' \subseteq E$ verifying:

$$\forall e \in E, e' \in E', \exists (n, m) \in \mathbb{N}^2 \mid c_\chi(e) - c_\chi(e') = n + \sqrt{-1} \cdot m \vee c_\chi(e') - c_\chi(e) = n + \sqrt{-1} \cdot m \text{ [P]}$$

and it is unique.

Proof. Let us consider two sets E'_1 and E'_2 so that $E'_1 \cap E'_2 = \emptyset$. If those exist, then one can check that the above property [P] only holds if $\forall (e'_1, e'_2) \in E'_1 \times E'_2$, $Re(c_\chi(e'_1)) = Re(c_\chi(e'_2))$ and $Im(c_\chi(e'_1)) = Im(c_\chi(e'_2))$. Thus $c_\chi(e'_1) = c_\chi(e'_2)$. let us denote this value c .

By definition of c_χ , two edges e'_1 and e'_2 sharing the same connectivity factor cannot be separated by an edge whose summit or end has an in- or out-exogenous degree different to zero. Thus, the connectivity factor of any edge e'_3 located between e'_1 and e'_2 in χ is equal to c .

Thus, the set E' made out of the union of E'_1 , E'_2 and all the edges between any pair of nodes of E'_1 and E'_2 still verifies [P].

By applying this to any set verifying [P], it follows that there is a maximal set verifying [P], and that it is unique.

Definition 9.10: Non ambiguous cell weighting. Be $G = (V_G, E_G)$ a cast. Be $\chi^- = (V, E)$ a partially independent cell of G . Be $[\chi^-/[Temp.\chi^-]]$ the expression of χ^- over its own, finite template.

By definition, χ^- being partially independent, then $\exists!$ maximal set $E' \subseteq E \mid \forall e \in E, e' \in E'$, tels que $e' \in follow_\chi(e) \vee e \in follow_\chi(e')$, $\exists (n, m) \in \mathbb{N}^2 \mid c_\chi(e) - c_\chi(e') = n + \sqrt{-1} \cdot m$.

We define the weighting operator $pond$ that associates, to $[\chi^-/[Temp.\chi^-]]$, the restriction of this matrix to E' :

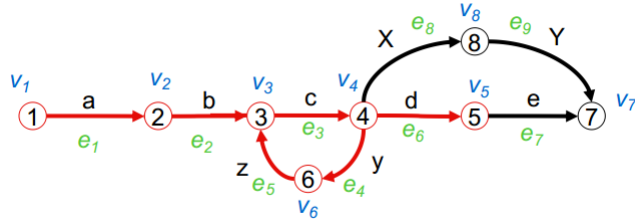
$$\begin{aligned} [pond.[\chi^-/[Temp.\chi^-]]_{i,j}] &= [[\chi^-/[Temp.\chi^-]]_{i,j}] \text{ if } \exists e \in E' \text{ characterized by the } j^{th} \\ &\quad \text{triple of } \mathcal{T} \times \mathcal{L} \times \mathcal{T} \\ &= [0] \text{ else.} \end{aligned}$$

Definition 9.11: Mod for non ambiguous schematic cells. Be G a cast, be χ^- a partially independent cell of G and χ^+ a cell so that $root(\chi^+) = root(\chi^-)$, $leaf(\chi^+) = leaf(\chi^-)$ and so that no other node of χ^+ belongs to G . This triple defines a non-ambiguous modification Mod .

The template of the graph H resulting from this modification is:

$$[H/[Temp.H]]^\infty = [G/[Temp.G]]^\infty - pond.[\chi^-/[Temp.\chi^-]]^\infty + [\chi^+[Temp.\chi^+]]^\infty$$

Example. Let us consider the following schema S , that has been introduced previously:



Let us consider that we want to replace the highlighted cell χ^- by a single edge χ^+ labelled Ω .

The representation of the three graphs over their respective, own finite templates is:

$$[S/[Temp.S]] = \begin{matrix} & a & b & c & d & e & y & z & X & Y \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix} & \left[\begin{array}{cccccccccc} [v_1] & [0] & [0] & [0] & [0] & [0] & [0] & [0] & [0] & [0] \\ -[v_2] & [v_2] & [0] & [0] & [0] & [0] & [0] & [0] & [0] & [0] \\ [0] & -[v_3] & [v_3] & [0] & [0] & [0] & [0] & -[v_3] & [0] & [0] \\ [0] & [0] & -[v_4] & [v_4] & [0] & [v_4] & [0] & [0] & [v_4] & [0] \\ [0] & [0] & [0] & -[v_5] & [v_5] & [0] & [0] & [0] & [0] & [0] \\ [0] & [0] & [0] & [0] & [0] & -[v_6] & [v_6] & [0] & [0] & [0] \\ [0] & [0] & [0] & [0] & -[v_7] & [0] & [0] & [0] & [0] & -[v_7] \\ [0] & [0] & [0] & [0] & [0] & [0] & [0] & [0] & -[v_8] & [v_8] \end{array} \right] \end{matrix}$$

$$[\chi^-/[Temp.\chi^-]] = \begin{matrix} & a & b & c & d & y & z \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{matrix} & \left[\begin{array}{cccccc} [v_1] & [0] & [0] & [0] & [0] & [0] \\ -[v_2] & [v_2] & [0] & [0] & [0] & [0] \\ [0] & -[v_3] & [v_3] & [0] & [0] & -[v_3] \\ [0] & [0] & -[v_4] & [v_4] & [v_4] & [0] \\ [0] & [0] & [0] & -[v_5] & [0] & [0] \\ [0] & [0] & [0] & [0] & -[v_6] & [v_6] \end{array} \right] \end{matrix}$$

$$[\chi^+[Temp.\chi^+]] = \begin{matrix} & \Omega \\ \begin{matrix} 1 \\ 5 \end{matrix} & \left[\begin{array}{c} [v_1] \\ -[v_5] \end{array} \right] \end{matrix}$$

In a previous example, we calculate the connectivity factors of the edges of χ^- , page 233: the conclusion is that χ^- is partially independent, with $E' = \{e_6\}$.

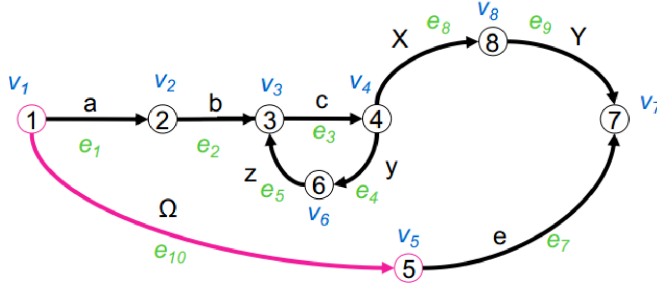
Given e_6 is characterized by the triple $(4, d, 6)$ in $\mathcal{T} \times \mathcal{L} \times \mathcal{T}$, it follows that:

$$pond.[\chi^-/[Temp.\chi^-]] = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} a \\ b \\ c \\ d \\ y \\ z \end{array} \begin{bmatrix} [0] & [0] & [0] & [0] & [0] & [0] \\ [0] & [0] & [0] & [0] & [0] & [0] \\ [0] & [0] & [0] & [0] & [0] & [0] \\ [0] & [0] & [0] & [v_4] & [0] & [0] \\ [0] & [0] & [0] & -[v_5] & [0] & [0] \\ [0] & [0] & [0] & [0] & [0] & [0] \end{bmatrix}$$

Hence, by denoting $[H/[Temp.H]]^\infty$ the matrix:
 $[S/[Temp.S]]^\infty - pond.[\chi^-/[Temp.\chi^-]]^\infty + [\chi^+/[Temp.\chi^+]]^\infty$,
 we have:

$$[H/[Temp.H]] = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \begin{array}{c} a \\ b \\ c \\ e \\ y \\ z \\ X \\ Y \\ \Omega \end{array} \begin{bmatrix} [v_1] & [0] & [0] & [0] & [0] & [0] & [0] & [0] & [v_1] \\ -[v_2] & [v_2] & [0] & [0] & [0] & [0] & [0] & [0] & [0] \\ [0] & -[v_3] & [v_3] & [0] & [0] & [0] & -[v_3] & [0] & [0] \\ [0] & [0] & -[v_4] & [0] & [v_4] & [0] & [0] & [v_4] & [0] \\ [0] & [0] & [0] & [v_5] & [0] & [0] & [0] & [0] & -[v_5] \\ [0] & [0] & [0] & [0] & -[v_6] & [v_6] & [0] & [0] & [0] \\ [0] & [0] & [0] & -[v_7] & [0] & [0] & [0] & -[v_7] & [0] \\ [0] & [0] & [0] & [0] & [0] & [0] & -[v_8] & [v_8] & [0] \end{bmatrix}$$

This matrix represents the following graph, that does match what shall be expected from the modification: the annotation sequences implying the execution of any root-to-leaf path of χ^- are made impossible, due to the disappearance of the edge d ; no other word from the language of S , other than those implying the execution of χ^- , is impacted by this modification.



9.3.3.4 Mod for Ambiguous Schematic Cells

Ambiguous cells are such that there is no set of edges E' containing edges of the cell that shall be involved in all, and only in, the root-to-leaf paths of the cell, contrary to partially independent cells. In partially independent cells, deleting those edges prevents from keeping words resulting from the execution of the cell, while maintaining all the other edges of the cell (that participate in defining exogenous words) – which is exactly the behaviour we expect from the *Mod* operator.

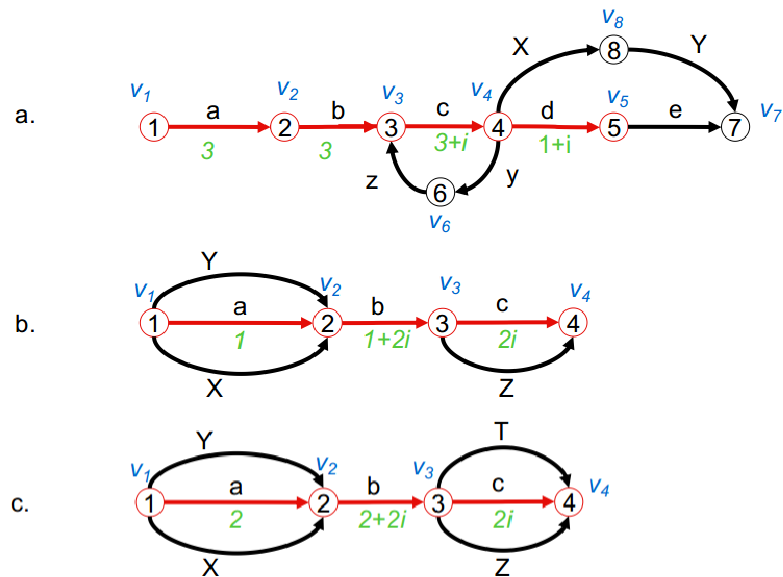
It follows that while side effects can be avoided in case of partially independent cells, there will always be a collateral loss of vocabulary when defining a *Mod* modification with an ambiguous cell as input.

It will also happen that there might be not only one, but several possible ways to perform a *Mod* operation with an ambiguous cell, yielding graphs differing one from the others by the words not implying the execution of a root-to-leaf path of the cell that are lost.

The connectivity factor presented above can be used to evaluate, for each solution, the extent of the collateral effect, based on a local measure of the connectivity of each edge of the cell to the rest of the graph – which provides a rough indication of the extent of the collateral effects resulting from deleting the edges in question. One strategy, for the definition of *Mod* in case of an ambiguous cell could have been to systematically go for the, if it is unique, solution that eliminates the edges of the cell whose connectivity factor is minimal. Yet, there may be more than one such ‘minimal’ solution: defining a deterministic operator shall thus either require further heuristics, or some arbitrary rule, that may be irrelevant from an editorial point of view; moreover, the minimal solution is based on a local evaluation of the connectivity of the edges, that may not reflect the number of words each edge of the cell actually, when considering the whole graph, be involved in; eventually, from the point of view of the editor who defines the *Mod* operation, the best solution may be different from any minimal one.

Thus the strategy we propose provides the user with a variety of solutions, ranked from the minimal ones, as defined above, to solutions with more local side effects – and it is up to her to chose among those solutions, based on personal and intellectual motives.

Let us enlighten this proposition by means of three examples. In the following graphs, ambiguous cells are highlighted in red; green values represent the connectivity factor of each edge of those cells.



Let us make the assumption that, in any case, the editor wants to replace the red cell by a single edge labelled Ω .

- a. After the modification, the word $abcd(e)$ must not be part of the graph anymore; yet, because the cycle cyz is not part of the cell, then it shall mean that the words $ab(cyz)^*cd(e)$ shall be kept – otherwise, their loss shall be reagrded as a side-effect of the modification. The modification is thus paradoxical, since the word to ban is also one of the words to maintain.
- b. In this case, after the modification, the word abc shall not be part of the language of the graph. This implies, as we have seen in the previous paragraph, that at least one of the three edges a , b or c shall be deleted. It is quite clear though that deleting any of those edges will have side effects; it also appears the extent of the side effects, for each edge, is not the same: deleting a implies losing abZ , that is not a word implying the execution of the cell; deleting b results in the loss of five such words (Ybc , YbZ , abZ , Xbc and XbZ); deleting c in the loss of two (Ybc and Xbc). Here, it is thus possible to rank the three solutions by the quantitative extent of their side effects.
- c. In this case, contrary to b., deleting a or c implies the loss of an equal number of words.

Let us now consider this problem from a more formal point of view. Again, we consider here 'ambiguous) cells limited to one linear graph.

Definition 9.12 : Decomposition of an ambiguous cell into equi-connected zones. Be $G = (E_G, V_G)$ a cast, $\chi = (E, V)$ a cell. The decomposition into equi-connected zones of χ is defined as follows:

$$\exists I \in \mathcal{P}(\mathbb{N}) \mid E = \bigcup_{i \in I} \{E_i\},$$

where $\forall i \in I, E_i \subseteq E$ so that:

- $\forall e, e' \in E_i, c_\chi(e) = c_\chi(e')$;
- $\forall e, e' \in E, c_\chi(e) = c_\chi(e') \Rightarrow \exists i \in I \mid (e, e') \in E_i$.

For any i , let c_i denote the connectivity factor of any edge from E_i .

Definition 9.13 : Greatest common part of connectivity factors.

Be G a cast, χ a cell and $\{E_i\}_{i \in I}$ the decomposition into equi-connected zones of χ . The greatest common part of the connectivity factors of χ is the complex number c_χ^{co} defined by:

$$\forall i \in I, \exists (n, m) \in \mathbb{N}^2 \mid c_i - c_\chi^{co} = n + \sqrt{-1} \cdot m \quad [\text{P}]$$

and so that for all other complex number c' verifying vérifiant [P], $\exists (k, l) \in \mathbb{N}^2 \mid c_\chi^{co} - c = k + \sqrt{-1} \cdot m$.

Definition 9.14 : *Mod operator, general case.* Be $G = (V_G, E_G)$ a cast, $\chi^- = (V, E)$ a linear cell of G and χ^+ a cell so that $root(\chi^-) = root(\chi^+)$ and $leaf(\chi^-) = leaf(\chi^+)$, and sharing no other node with G . Be $\{E_i\}_{i \in I}$ the decomposition into equi-connected zones of χ^- and $c_{\chi^-}^{co}$ the greatest common part of the connectivity factors of χ .

1. Be $I_1 = \{i_{1,j}\} \subset I$ the set of natural numbers so that: $\forall j \in I_1, \forall e \in E_{i_{1,j}}$,

$$\begin{aligned} Re(c_{\chi^-}(e) - c_{\chi^-}^{co}) + Im(c_{\chi^-}(e) - c_{\chi^-}^{co}) = \\ \min_{e' \in E} (Re(c_{\chi^-}(e) - c_{\chi^-}^{co}) + Im(c_{\chi^-}(e) - c_{\chi^-}^{co})) \end{aligned}$$

Then the primary graph propositions for the *Mod* modification with G , χ^- and χ^+ are the $|I_1|$ graphs matching the definition of H below:

$$[H/[Temp.H]]^\infty = [G/[Temp.G]]^\infty - pond.[\chi^-/[Temp.\chi^-]]^\infty + [\chi^+[Temp.\chi^+]]^\infty$$

where:

$$\begin{aligned} [pond.[\chi^-/[Temp.\chi^-]]_{k,l}] &= [[\chi^-/[Temp.\chi^-]]_{k,l}] \text{ if } \exists e \in E_{i_{1,j}} \\ &\text{characterized by the } l^{\text{th}} \text{ triple from } \mathcal{T} \times \mathcal{L} \times \mathcal{T} \\ &= [0] \text{ else.} \end{aligned}$$

with j varying in I_1 .

2. Be $I_2 = \{i_{2,j}\} \subset I \setminus I_1$ the set of natural numbers so that: $\forall j \in I_2, \forall e \in E_{i_{2,j}}$,

$$\begin{aligned} Re(c_{\chi^-}(e) - c_{\chi^-}^{co}) + Im(c_{\chi^-}(e) - c_{\chi^-}^{co}) = \\ \min_{e' \in E \setminus \bigcup_{j \in I_1} E_{i_{1,j}}} (Re(c_{\chi^-}(e) - c_{\chi^-}^{co}) + Im(c_{\chi^-}(e) - c_{\chi^-}^{co})) \end{aligned}$$

Then the primary graph propositions for the *Mod* modification with G , χ^- and χ^+ are the $|I_2|$ graphs matching the definition of H below:

$$[H/[Temp.H]]^\infty = [G/[Temp.G]]^\infty - pond.[\chi^-/[Temp.\chi^-]]^\infty + [\chi^+[Temp.\chi^+]]^\infty$$

where

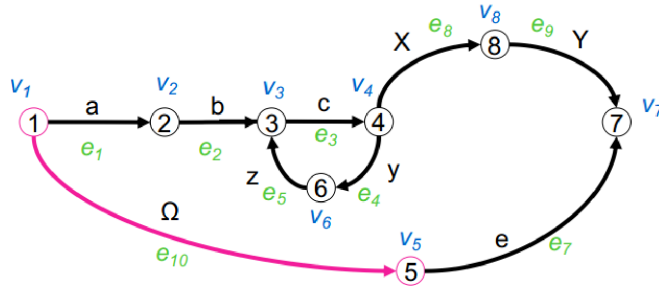
$$\begin{aligned} [pond.[\chi^-/[Temp.\chi^-]]_{k,l}] \\ &= [[\chi^-/[Temp.\chi^-]]_{k,l}] \text{ if } \exists e \in E_{i_{2,j}} \\ &\text{characterized by the } l^{\text{th}} \text{ triple from } \mathcal{T} \times \mathcal{L} \times \mathcal{T} \\ &= [0] \text{ sinon.} \end{aligned}$$

with j varying in I_2 .

3. Etc.

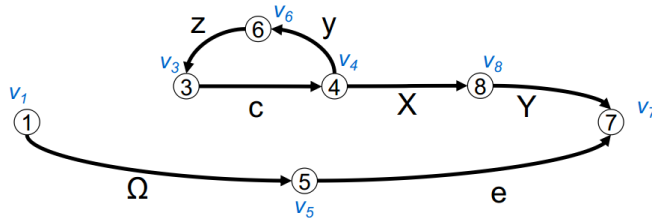
Nota Bene. Among the set of graphs yielded by *Mod*, the choice can be left to the user – except when $\exists i \in I, |c_i - c_{\chi^-}^{co}| = 0$, in which case the primary graph is unique, which would mean that the cell is actually partially independent.

Exemple. Let us go back to the first of the three examples introduced on page 239. In this case, the the greatest common part of the connectivity factors of χ^- is zero. Three equi-connected zones can be identified: $E_1 = \{a, b\}$; $E_1 = \{c\}$; $E_1 = \{d\}$. The one for which the sum of the real and the imaginary parts of the connectivity factor is minimal is $E_3 : I_3 = \{3\}$. Thus, only one primary graph will be defined: it will be obtained by deleting the edge d and by inserting χ^+ adequately, which gives:



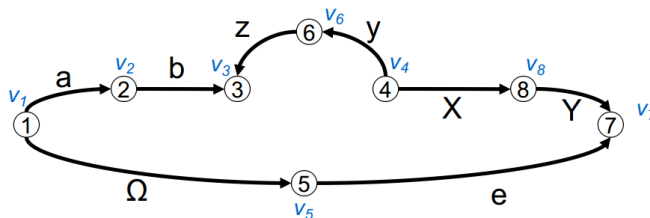
Noteworthy, the resulting graph is the one that we got at the end of Paragraph 9.3.3.3 page 238, where the cell to be replaced differed slightly from χ^- , in that it incorporated the cycle cyz and, for that reason, was partially independent: this suggests that the graph that minimizes the side effects is precisely the one we get when calculating the graph with the smallest partially independent cell χ^- is included in.

If we calculate the secondary graphs: $I_2 = \{1\}$. Then $pond.[\chi^-/[Temp.\chi^-]]$ with I_2 is a subgraph of χ^- containing only the edges a and b . The unique secondary graph for this modification is thus:



We can notice that the cast we get does not respect the SeAG model: such a graph shall then only be considered as the basis for further modifications. Still, it is a possible interpretation of the modification described above.

The tertiary graph, even more baroque, is the following:



9.3.3.5 Notation

We have thus defined how to calculate one or more graphs resulting from the substitution of a cell of a cast by another cell, regardless of the nature of the cell to be replaced (independent, partially independent and ambiguous). Because this definition is homogeneous across the different kinds of cells, we can introduce a unique syntax for declaring such modifications.

Definition 9.14 : Substitution transformations. Be a cast G , be $\chi^- \subseteq S$ a schematic cell and χ^+ another cell possessing the same root and leaf as χ^- , and sharing no other node with G .

Let us denote $Mod(\chi^-; \chi^+).G$ the ordered list of graphs resulting from the modification described above.

9.4 Split Operator

The above *Mod* operator offers a means to amend a cast subgraph by subgraph, by replacing, more or less, a given cell and inserting, at the same place (that is, between the same pair of root and leaf) a substitution graph.

We now turn to the *Split* operation that apply to a single node and enables to double it and to draw an ϵ edge between the two newly created nodes. In the next paragraph, we introduce *Unite*, that acts the other way round and merges two nodes connected by an edge.

Notation. Be $[M]$ the expression of a graph on an infinite template. It is a block matrix, whose block-lines and block-columns are indexed over \mathcal{T} et $\mathcal{T} \times \mathcal{L} \times \mathcal{T}$ respectively, and whose inner blocks are indexed over \mathcal{I}^2 .

Be i, j, x, y indexes over \mathcal{T} , $\mathcal{T} \times \mathcal{L} \times \mathcal{T}$, \mathcal{I} and \mathcal{I} respectively. Then $[[M]_{i,j}]$ denotes the inner block matrix of $[M]$ characterized by the type value t of index i in \mathcal{T} , and by the triple (t, l, t') of index j in $\mathcal{T} \times \mathcal{L} \times \mathcal{T}$. Naturally, $[[M]_{i,j}]_{x,y}$ denotes the coefficient of index (x, y) in that inner matrix.

Definition 9.4.15 : Cod operator. Be $[M]$ the expression of a graph on an infinite template. We define the *Cod* operator as a means to replace all the nodes of a graph whose type has a given value, by new nodes of the same type:

Be i so that $\exists j \mid [[M]_{ij}] \neq [0]$. Then be $Id_i^V = \{x \text{ so that } \exists j, y \mid [[M]_{ij}]_{xy} \neq 0\}$.

The operation of the *Cod* operator to the i_1^{th} block-line of $[M]$, denoted $Cod(i_1).[M]$, is defined as follows:

$$\begin{aligned} [[Cod(i_1).[M]]_{i,j}]_{x,y} &= [[M]_{i,j}]_{x,y} \text{ si } i \neq i_1 \\ &= [[M]_{i,j}]_{\sigma_1^{-1}(x),y} \text{ si } i = i_1 \quad \wedge \quad \exists x' \mid [[M]_{i,j}]_{x',y} \neq 0 \quad \wedge \quad x = \sigma_1(x') \\ &= 0 \text{ else,} \end{aligned}$$

where σ_1 is a permutation of \mathbb{N} verifying:

1. $\forall n \in \bigcup_{i \in \mathbb{N}, i \neq i_1} Id_i^V \setminus Id_{i_1}^V, \sigma_1(n) = n$;

$$2. \forall n_1 \in Id_{i_1}^V, \sigma_1(n_1) \notin \bigcup_{i \in \mathbb{N}} Id_i^V.$$

Example. Be the following block-matrix:

$$[M] = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{cccc} a & b & c & \epsilon \\ \left[\begin{array}{cccc} [v_1] & [v_1] & [0] & [0] \\ [0] & [0] & [0] & [0] \\ [0] & [0] & -[v_3] & [0] \\ -[v_2] & -[v_2] & [0] & [v_2] \\ [0] & [0] & [v_2] & [v_2] \end{array} \right] \end{array}$$

This matrix cannot represent a well-formed eAG or SeAG, since more than one type value are associated with the node v_2 : the node appears both on the 4th and 5th line-blocks of $[M]$. Let us apply Cod to $[M]$ with 4 as an argument.

We can see that $\bigcup_{i \in \mathbb{N}, i \neq i_1} Id_i^V \setminus Id_{i_1}^V = \{1; 3\}$ and $\bigcup_{i \in \mathbb{N}} Id_i^V = \{1; 2; 3\}$. A permutation of \mathbb{N} verifying the two conditions above is the following:

$$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 4 & 3 \end{pmatrix}$$

This permutation can then be used to determine the coefficients of the inner-matrices of the line 4 of $Cod(4).[M]$. As an illustration:

$$[[Cod(4).[M]]_{4,1}]_{4,1} = [[M]_{4,1}]_{2,1} = -1$$

which means, synthetically, that $[[Cod(4).[M]]_{4,1}] = -[v_4]$.

Definition 9.4.16 : Split operator. Be $G = (V, E)$ a cast.

Be $T_G = \{t \in \mathcal{T}; \exists v \in V \mid type(v) = t\}$.

Be $[G/[Temp.G]]^\infty$ the representation of G on its own template. We define the *Split* operator, taking two arguments $t_0 \in \mathcal{T}, t_1, t_2 \in \mathcal{T} \setminus T_G$, by:

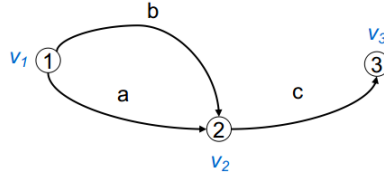
1. If $t_0 \notin T_G$, then $[Split(t_0, t_1, t_2).[G/[Temp.G]]^\infty] = [G/[Temp.G]]^\infty$.
2. Else: be i_0 the index of t_0 in \mathcal{T} ; be i_1 and i_2 the respective indexes of $t_1, t_2 \in \mathcal{T}$. Be j_ϵ the index of the triple (i_1, ϵ, i_2) in $\mathcal{T} \times \mathcal{L} \times \mathcal{T}$. Then :

$$\begin{aligned} & [Split(t_0, t_1, t_2).[G/[Temp.G]]^\infty]_{k,l} \\ &= [[G/[Temp.G]]^\infty]_{k,l} \text{ if } k \notin \{i_0, i_1, i_2\} \wedge l \neq j_\epsilon \\ &= 0 \text{ if } k = i_0 \\ &= [[Cod(i_1).[G/[Temp.G]]^\infty]_{i_0,l}] \text{ if } k = i_1 \wedge l \neq j_\epsilon \wedge [[G/[Temp.G]]^\infty]_{i_0,l} < [0] \\ &= [[Cod(i_2).[G/[Temp.G]]^\infty]_{i_0,l}] \text{ if } k = i_2 \wedge l \neq j_\epsilon \wedge [[G/[Temp.G]]^\infty]_{i_0,l} > [0] \\ &= [A] \text{ if } (k, l) = (i_1, j_\epsilon) \\ &= [B] \text{ if } (k, l) = (i_2, j_\epsilon) \end{aligned}$$

where:

- $[A] = - \sum_{j \in J_-} [[Cod(i_1).[G/[Temp.G]]]_{i_0,j}^\infty]$,
where $\{[[Cod(i_1).[G/[Temp.G]]]_{i_0,j}^\infty]\}_{j \in J_-}$ is the set of all the negative inner matrices from the i_0^{th} block-line of $[G/[Temp.G]]^\infty$, whose coefficients have been changed⁴ by the operator Cod ;
- $[B] = - \sum_{j \in J_+} [[Cod(i_2).[G/[Temp.G]]]_{i_0,j}^\infty]$,
where $\{[[Cod(i_2).[G/[Temp.G]]]_{i_0,j}^\infty]\}_{j \in J_+}$ is the set of all the positive inner matrices from the i_0^{th} block-line of $[G/[Temp.G]]^\infty$, whose coefficients have been changed by the operator Cod .

Example. Be the following cast S :



Its representation over its own template is:

$$[S/[Temp.S]] = \begin{matrix} & & a & b & c \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \begin{bmatrix} [v_1] & [v_1] & [0] \\ -[v_2] & -[v_2] & [v_2] \\ [0] & [0] & -[v_3] \end{bmatrix} \end{matrix}$$

Let us consider that we need to split the node of type 2. According to the nomenclature of the definition above, we have the following items:

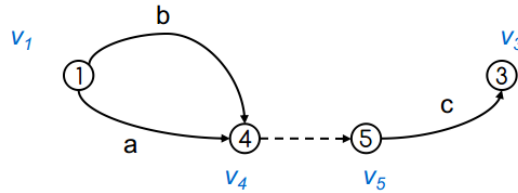
- the set of types of S is $T_S = \{1; 2; 3\}$;
- thus three possible indexes in \mathcal{T} so that two of them do not belong to T_S are $i_0 = 2$, $i_1 = 4$ and $i_2 = 5$: the node of type 2 will then be replaced by a pair of nodes of types 4 and 5;
- $J_- = \{1; 2\}$ (those are the indexes of the negative inner matrices of the block-line i_0 in $[S/[Temp.S]]$) and $J_+ = \{3\}$;
- see the example page 244 for the definition of two possible permutations for $Cod(4)$ are $Cod(5)$;
- the matrix $[A] = -([Cod(4).[S/[temp.S]]]_{2,1} + [Cod(4).[S/[temp.S]]]_{2,2}) = -([-v_4] + [-v_4]) = [v_4]$;
- the matrix $[B] = -[Cod(5).[S/[temp.S]]]_{2,3} = -[v_5]$;

⁴... so that one node shall not be attributed two different type values.

- the block matrix of $Split(2, 4, 5).S$ is thus:

$$[Split(2, 4, 5).[S/[Temp.S]]] = \begin{matrix} & \begin{matrix} a & b & c & \epsilon \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} [v_1] & [v_1] & [0] & [0] \\ [0] & [0] & [0] & [0] \\ [0] & [0] & -[v_3] & [0] \\ -[v_4] & -[v_4] & [0] & [A] \\ [0] & [0] & [v_5] & [B] \end{bmatrix} \end{matrix}$$

This matrix represents the following graph:



Notation. We will denote $Split(i, j, k).G$ the graph obtained by the operation of $Split$ on G with i, j, k as parameters.

9.5 Unite Operator

The last kind of modification we define, denoted $Unite$. Its behaviour is somehow symmetrical to $Split$, since it is meant to merge two nodes connected by an ϵ edge, under certain conditions.

9.5.1 Situations where the $Unite$ Operator Applies

Intuitively speaking, given two nodes v_1 and v_2 from a graph G , so that there is a unique edge ϵ so that $v_1[\epsilon]v_2 \subseteq G$, then the effect of $Unite$ operated on the nodes v_1 and v_2 will be a graph in the pattern $v_1[e]v_2$ will be replaced by a single node v , so that v shall be the new summit of all the edges v_1 or v_2 were the summit of in G , e excluded, and so that v shall be the end of all the edges v_1 or v_2 were the end of in G , e excluded. Indeed, the application of $Unite$ should, ideally, leave the annotation language of the graph untouched, so all the other label sequences involving either v_1 or v_2 shall be restored in the image of G by $Unite$, by using v as a surrogate for either v_1 or v_2 .

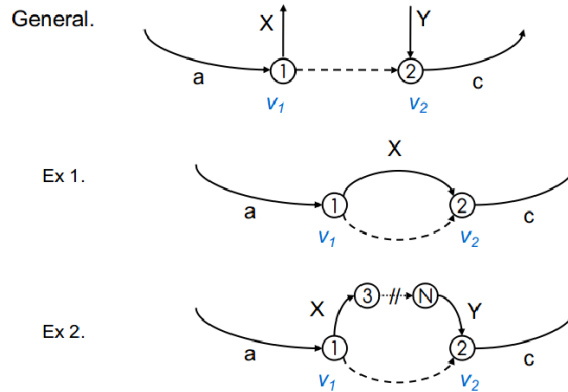
Yet, leaving the language of G untouched by the application of $Unite$ cannot be verified in certain cases for which the operator will not be defined.

Definition 9.16 : Situations where the $Unite$ operator applies. Be a cast G . Be two nodes v_1, v_2 of G so that $\exists e \mid v_1[e]v_2 \subseteq G$. The couple $(v_1; v_2)$ is eligible for the application of $Unite$ iff:

1. $label(e) = \epsilon$;
2. $\nexists e' \mid v_2[e']v_1 \subseteq G$;
3. $out(v_1) = 1$ or $in(v_2) = 1$;
4. $\nexists v, e, e' \mid v[e]v_1 \subseteq G \wedge v[e']v_2 \subseteq G \wedge label(e) = label(e')$
 AND $\nexists v, e, e' \mid v_1[e]v \subseteq G \wedge v_2[e']v \subseteq G \wedge label(e) = label(e')$

Justification. Let us explain the rationale behind each item:

1. We restrict the label of the edge between v_1 and v_2 to ϵ in order to stick to our target, that is, to make *Unite* conservative regarding the annotation language of the graph. If two nodes connected together by an edge labelled otherwise, first, the user shall change the label of that edge by means of the *Mod* operator.
2. Merging two nodes that do not respect that condition shall result in creating a loop, which cannot be represented by means of an incidence matrix.
3. The third condition prevents from merging the two nodes v_1 and v_2 in the following cases:



The two particular cases Ex.1 and Ex.2, deriving from the general case, can be described as follows:

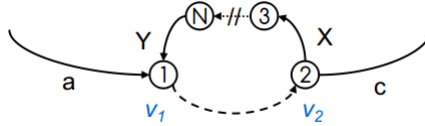
- As for the item 2. above, merging v_1 and v_2 in the case of Ex.1 would result in creating a loop. More generally, the restrictive item 3. from Definition 9.16 imposes the following condition on v_1 and v_2 :

$$\exists e, e' \mid v_1[e']v_2 \wedge v_1[e]v_2 \Rightarrow e' = e;$$

In the case where v_1 and v_2 would be connected together by several edges, would it be necessary to unite both nodes nonetheless, the user could replace the set of edges between the two nodes by one epsilon edge by means of the *Mod* operator first.

- The restrictive item 3. from Definition 9.16 also discards certain patterns, illustrated by the Ex.2. above, that is, the situation where there is a path leading from v_1 to v_2 in parallel of the ϵ edge to be deleted. Indeed, merging the two nodes would then change this parallel path into a cycle – which would impact the annotation language of the graph greatly. We will see, in Paragraph 10.7.2, how to make an existing label sequence in a schema cyclic.
- In general, if there is another edge than ϵ going out of v_1 AND another edge than ϵ ending on v_2 , merging the two nodes might not create any cycle in the graph – to be sure, the graph would have to be crawled through, in order to check if the two edges belong to a common path or not – but will result, in any case, in a substantial modification of the annotation language of the graph: while the original graph does not enable Y to be followed by X , in a graph where v_1 and v_2 are merged into one node, the sequence YX will be allowed.

Still, the following situation is not discarded by the restriction number 3:



4. The restrictive item 4. from Definition 9.16 cannot be explained right now: it is necessary, still, for propagating the *Unite* operation to the instances of the cast it is defined onto. See Paragraph 10.5 page 280.

9.5.2 *Unite* Operator Definition

Definition 9.17: *Unite* operator. Be $G = (V_G, E_G)$ a cast. Be \mathcal{I}_G the set of the identifiers of the nodes of G , and \mathcal{T}_G the set of type values of those nodes.

Be two nodes $(v_a, v_b) \in V_G^2$ so that $type(v_a) = t_a$, $type(v_b) = t_b$, $id(v_a) = a$, $id(v_b) = b$, so that the pair of nodes verifies the conditions given in Definition 9.16.

Be also J the index of the triple (t_a, ϵ, t_b) in $\mathcal{T} \times \mathcal{L} \times \mathcal{T}$.

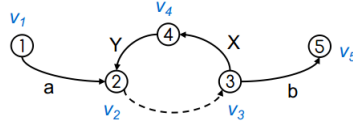
Be then a node v_N so that $id(v_N) \notin \mathcal{I}_G$ and so that $t_N = type(v_N) \notin \mathcal{T}_G$.

Then the modification *Unite* operated on G , with t_a , t_b and t_N as arguments, is defined

as follows:

$$\begin{aligned}
& [[Unite(t_a, t_b, t_N).[G/[Temp.G]]]_{i,j}]_{x,y} \\
& = 0 \text{ if } j = J \\
& \text{(the } \epsilon \text{ edge is deleted from } G\text{)} \\
& = 0 \text{ if } i \in \{t_a, t_b\} \\
& \text{(the lines corresponding to } t_a \text{ and } t_b \text{ are reset to zero)} \\
& = [[G/[Temp.G]]_{i,j}]_{x,y} \text{ if } i \notin \{t_a, t_b, t_N\} \wedge j \neq J \\
& \text{(the lines corresponding to the other type values of } \mathcal{T}_G \text{ are left intact)} \\
& = [[G/[Temp.G]]_{t_a, j_a}]_{a,y} + [[G/[Temp.G]]_{t_b, j_b}]_{b,y} \text{ if } i = t_N \wedge x = N \\
& \quad \wedge j \neq J, \\
& \text{(for each } j \text{ corresponding to a triple } (t, l, t'), \text{)} \\
& \text{(} j_a \text{ corresponds to } (t, l, t_a) \text{ or } (t_a, l, t')\text{)} \\
& \text{(and } j_b \text{ corresponds to } (t, l, t_b) \text{ or } (t_b, l, t')\text{ :)} \\
& \text{(this line means that } v_a \text{ and } v_b, \text{ as summits and ends of an edge)} \\
& \text{(different from the } \epsilon \text{ edge to be merged, are replaced by } v_N\text{)} \\
& = 0 \text{ if } i = t_N \wedge x \in \{a, b\} \\
& \text{(the nodes } v_a \text{ and } v_b \text{ are deleted here)}
\end{aligned}$$

Example. Be the following cast G and its template-based representation:

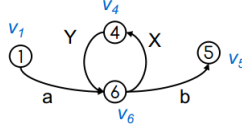


$$[G/[Temp.G]] = \begin{matrix} & a & \epsilon & X & Y & b \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} [v_1] & [0] & [0] & [0] & [0] \\ -[v_2] & [v_2] & [0] & -[v_2] & [0] \\ [0] & -[v_3] & [v_3] & [0] & [v_3] \\ [0] & [0] & -[v_4] & [v_4] & [0] \\ [0] & [0] & [0] & [0] & [v_5] \end{bmatrix} \end{matrix}$$

The pair of nodes (v_2, v_3) verifies the conditions given in Definition 9.16. The operation $Unite(2, 3, 6)$ can thus be performed on S . With the nomenclature from the Definition 9.17, $J \equiv 2$, $t_N \equiv 6$ and $v_N \equiv v_6$, $t_a \equiv 2$ and $v_a \equiv v_2$, $t_b \equiv 3$ and $v_b \equiv v_3$. Thus, as an illustration:

$$\begin{aligned}
[[Unite(t_a, t_b, t_N).[G/[Temp.G]]]_{6,1}]_{6,1} &= [[G/[Temp.G]]_{2,1}]_{2,1} + [[G/[Temp.G]]_{3,1}]_{3,1} \\
&= -1 + 0 \\
\Rightarrow [[Unite(t_a, t_b, t_N).[G/[Temp.G]]]_{6,1}] &= -[v_6]
\end{aligned}$$

In fine, the modification yields the following cast:



9.6 Operators Composability

We have defined three operators *Mod*, *Split* and *Unite*. Those operations are meant to be composed, in order for the user to define a transformation, producing a schema out of another schema. We establish here that the operators are indeed composable, in the sense that each of them produces a cast out of a cast – provided the following definitional properties to the definition of the *Mod* and *Split* operators, in order to take into account some limit cases that the above definitions of the operators do not consider:

Definition 9.18 : *Mod and Split extension to limit cases.* Be $G = (V_G, E_G)$ a cast.

1. Be χ a cell so that $\chi \not\subseteq G$. Be χ^+ any cell.
Then $Mod(\chi, \chi^+).G = G$.
2. Be χ a cell. Be χ^+ another cell so that $(root(\chi^+), leaf(\chi^+)) \neq (root(\chi), leaf(\chi))$.
Then $Mod(\chi, \chi^+).G = G$.
3. Be χ a cell. Be χ^+ a cell possessing a node v so that $v \notin \{root(\chi^+), leaf(\chi^+)\}$, and so that $\exists v' \in V_G$ so that $type(v) = type(v')$.
Then $Mod(\chi, \chi^+).G = G$.
4. Be χ a cell and χ^+ another cell so that $\exists e \mid root(\chi^+)[e]leaf(\chi^+) \subseteq \chi^+$, and so that $\exists e' \in E_G \mid root(\chi^+)[e']leaf(\chi^+) \subseteq G$ and $label(e) = label(e')$.
Then $Mod(\chi, \chi^+).G = Mod(\chi, \chi^+ \setminus root(\chi^+)[e]leaf(\chi^+)).G$.
5. Be $T_G = \{t \in \mathcal{T} \mid \exists v \in V_G ; type(v) = t\}$. Be $t_k \in \mathcal{T}$ so that $t_k \notin T_G$.
Then $Split(k).G = G$.

Thus, the operations *Mod* and *Split* are defined for any argument⁵.

Now let us check that the image of any cast by any modification is a cast.

Theorem 9.1. Be G a cast.

1. Whatever χ, χ^+ schematic cells, $Mod(\chi, \chi^+).G$ is a cast.
2. Whatever $k \in \mathbb{N}$, $Split(k).G$ is a cast.

⁵*Unite*, a contrario, is not defined for some arguments, and its definition otherwise is given in the previous paragraph.

3. Whenever $Unite(t_a, t_b, t_N).G$ is defined, it is a cast.

Proof. We must establish that whatever the operation:

- a. The image of G does not possess a connected part restricted to a node.
- b. The image of G does not contain loops.
- c. The nodes of the image of G are identified by their type value, and the edges of the image of G , by the types of their summit and end together with their label.

The points a. and b. above are granted, for the image of G is expressed by means of an incidence matrix (a template-based representation, actually), which does not enable the representation of either loops or isolate nodes. Let us check the point c. holds:

Regarding the *Mod* operator :

It is clear that, regardless of the cells considered, $Mod(\chi, \chi^+).G \subseteq G \cup \chi^+$.

Let us suppose now that there are two nodes v, v' among the nodes of $Mod(\chi, \chi^+).G$ so that $type(v) = type(v')$. By definition of the cast G , $\forall (v_1, v_2) \in G$, $type(v_1) \neq type(v_2)$, and the same for the nodes of χ^+ , which is also a cast. Thus, the only remaining possibility is $v \in G \wedge v' \in \chi^+$ (or the other way round). This situation splits into three cases:

- If $v \notin G \cap \chi^+ = \{root(\chi^+); leaf(\chi^+)\}$, then χ^+ is a cell possessing a node $v \notin \{root(\chi^+), leaf(\chi^+)\}$ so that $\exists v' \in V_G$ so that $type(v) = type(v')$, which gives $Mod(\chi, \chi^+).G = G$ by definition.
- Else, we can possibly have $v = root(\chi^+)$, which implies that $v' = v$ since the nodes of χ^+ are identified by their type value.
- Eventually, we can have $v = leaf(\chi^+)$, which implies $v' = v$ for the same reason.

Thus in any case, the nodes of $Mod(\chi, \chi^+).G$ are identified by their type value.

To conclude, we may note that there cannot be two edges sharing the same label between a given pair of nodes in a cast, that is, neither in G nor in χ^+ . Thus, the only situation that might imply the existence of two edges characterized by the same triple from $\mathcal{T} \times \mathcal{L} \times \mathcal{T}$ in $G \cup \chi^+$ is the situation described in the item 4. in Definition 9.18. Since this situation is solved by not considering the edge e of χ^+ that is characterised by the same triple as some edge from G , $e \notin Mod(\chi, \chi^+).G$. By applying this resolution principle to all the edges of χ^+ that shall be problematic, we make sure that *Mod* does produce a cast out of a cast.

Regarding the *Split* operator:

We set the case $Split(i).G = G$ apart. Otherwise, let us consider the type value of the nodes from $Split(i).G \setminus G$. Consider the case where $\exists j \mid [[G/[Temp.G]]_{ij}] \neq [0]$. G being a cast, that is, a graph in which there is a bijection between the type and the identifier values for nodes, $\exists !x \mid \forall j, y, [[G/[Temp.G]]_{ij}]_{xy} \neq 0$. Then the matrix representing $Split(i).G$ is defined by replacing the inner matrices from the i^{th} block-line of $[G/[Temp.G]]$ onto two different lines corresponding to a type value that is not present in G and, thanks to the *Cod* operator, by changing the identifier of the nodes thus dispatched onto two block-lines. *Cod* is defined so that, after this operation, the

nodes from those two lines do not share their identifier value, maintaining the bijection between types and node identifiers.

To conclude, the only new edge from $Split(i).G$ is the unique ϵ edge connecting the new nodes resulting from the split. Hence the edges of the resulting graph are well identified by a triple in $\mathcal{T} \times \mathcal{L} \times \mathcal{T}$.

Regarding the *Unite* operator:

Eventually, the only new node of $Unite(t_a, t_b, t_N).G$ possesses a type value that is not possessed by any node of G .

Chapter 10

eAG Bidirectional Transformations

We have defined SeAG transformations as the composition of *Mod*, *Split* and *Unite* modifications, in order to produce, out of an original schema, a cast that, if it is well defined, is an amended schema. We now consider how bidirectional transformations, operating in the domain of the instances, can be derived from those SeAG transformations, and play the following double role in the system we propose:

1. **Initialization.** Provided an original schema S and a corresponding eAG I_S , and provided a modification g defining a schema $S' = g.S$, the forward transformation derived from g will enable to define an eAG $I_{S'}$ that shall be expressible on the template of S' and contain ‘as much of the information contained in I_S as possible’.
2. **Synchronization** From there on, any update performed on $I_{S'}$ will have to be propagated to I_S by means of the backwards modification derived from g , and conversely, any update performed on I_S will have to be propagated to $I_{S'}$ by means of the forward modification derived from g .

As for transformations, it also appears that an eAG transformation shall result from the composition of elementary, bidirectional modifications, for example each deriving from a SeAG modification (see Paragraph 10.1) Before considering how to derive such an eAG modification out of a SeAG modification, we first focus on the compositional properties eAG modifications shall verify; we then define the notion of instance update and propose a temporal model for the synchronization of two instances that are updated in parallel.

10.1 Composing eAG Modifications

The general strategy for the definition of an eAG transformation is the following: each modification in the domain of the schemas (and casts) shall be translated into

a bidirectional modification in the domain of the instances. Thus, a transformation between two instances corresponds to the composition of the modifications derived from the corresponding composition of modification in the domain of schemas:

$$\begin{array}{ccccccc}
 S & \xrightarrow{\text{modif}_1^{\text{sch.}}} & G_1 & \xrightarrow{\text{modif}_2^{\text{sch.}}} & G_2 & \dots & \xrightarrow{\text{modif}_N^{\text{sch.}}} & S' \\
 & \Downarrow & & \Downarrow & & \dots & \Downarrow & \\
 I_S & \xleftarrow{\text{modif}_1^{\text{inst.}}} & H_1 & \xleftarrow{\text{modif}_2^{\text{inst.}}} & H_2 & \dots & \xleftarrow{\text{modif}_N^{\text{inst.}}} & I_{S'}
 \end{array}$$

For $I_{S'}$ to be an instance of S' , it has to be expressible on the template of S' . Consistently with this fact, each H_k graph above shall be expressible on the template of the corresponding cast G_k . In other words, H_k will have to be defined by setting the value of the inner matrices of $[temp.G_k]$ so that the resulting graph represents a graph. This requires the following property to be verified by $[H_k/[temp.G_k]]$

1. $\forall i, i', j \mid \exists [A_{ij}] > [0] \wedge [B_{i'j}] < [0]$ inner matrices of $[H_i/[temp.G_i]]$, then $size_W.[A_{ij}] = size_W.[B_{i'j}]$, where $size_W$ denotes the function that gives the number of columns of a matrix.
2. Moreover, it must be verified that each node shall be associated one and only one type value, i.e. that:

$$[[H_k/[Temp.G_k]]_{i,j}]_{x,y} \neq 0 \Rightarrow \forall i' \neq i, [[H_k/[Temp.G_k]]_{i',j}]_{x,y} = 0.$$

Noteworthy, each of the above $modif_k^{\text{inst}}$ shall also play the double role attributed to the transformations in the introductory purpose of this chapter: initialize the next graph H_k (or $I_{S'}$) and synchronize each $H_k - H_{k+1}$ pair.

Those few constraints are the only ones we impose on the H_k – which thus belong to a class of graphs much wider than the eAGs. Throughout this chapter, we will illustrate what seems to be a property of the eAG transformations we propose, that is, the fact that the composition of a sequence of eAG modifications that derive from a sequence of SeAG modifications that transform a well-formed SeAG into a well-formed SeAG, also transform a well-formed eAG into a well-formed eAG. Yet this property is just an assumption at this stage of our research.

10.2 Definition and Temporal Model of Instance Update

Definition 10.1 : Instance update Be a schema-instance pair $S - I_S$; be g a SeAG transformation and S' the image of S by g .

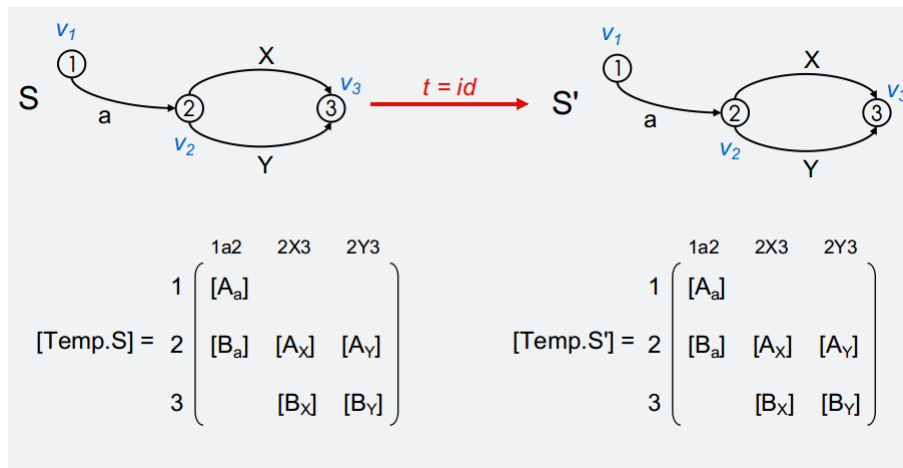
Let us consider that we have an instance $I_{S'}$ of S' so that $I_{S'}$ is the image of I_S by the eAG transformation g_I corresponding to g .

An update of either I_S or $I_{S'}$ is any change in the matrix values of $[I_S/[Temp.S]]$ or $[I_{S'}/[Temp.S']]$, so that the resulting matrix represents a well-formed eAG.

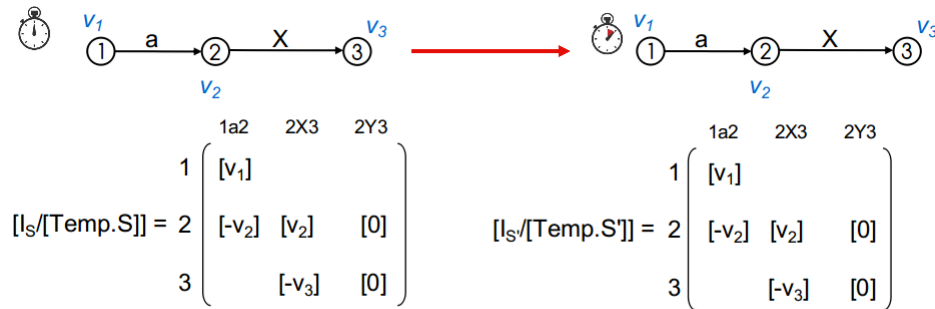
In a HCI-mediated annotation system, an update will correspond to the back-end consequences of an editorial intervention onto the primary corpus operated by the end

user, that is, one editor: identification of a passage and characterization of this passage with the definition of an element; definition of a comment, etc. Thus, the actions undertaken by the editor that will imply an update in the structured data will thus *take time*. The duration of an update can, in a collaborative and distributed setting, be the source of conflicts and/or ambiguities. To illustrate this aspect, consider the toy example below, in which two editors, for the sake of clarity, work on two schemas S and S' that are identical (in other words, the transformation of S into S' is the identity function). In such a case, it is clear that the corresponding eAG bidirectional transformation shall guarantee the identity between the instances.

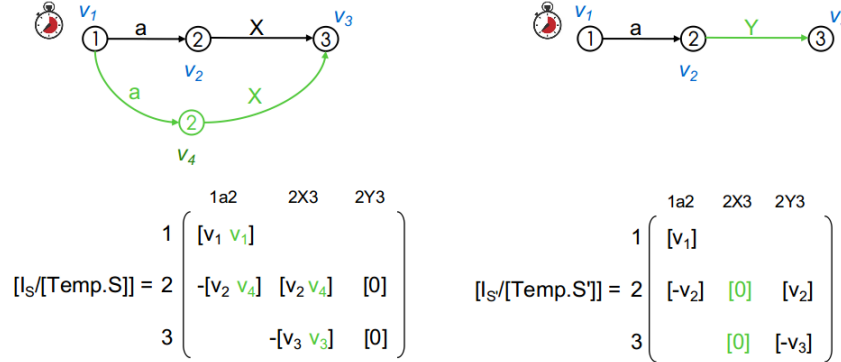
The template of S and S' is represented below: the matrices $[A_x]$ denote matrices that will take positive values in any expression over the corresponding template, and $[B_x]$, negative values.



Let us consider, as a starting point $t = t_0$, that there is a very elementary instance I_S of S in store, and the transformation that from S gives S' has just been defined. The determination of the eAG transformation corresponding to the SeAG transformation, and the calculation of I_S into data validated by S' takes some time too. At $t = t_1$, two instances I_S and $I_{S'}$ coexist.



Let us now consider that, from the moment t_1 , two editors are working, one on I_S and the other one on $I_{S'}$, and modify each instance as follows, for example before committing:



Based on this situation, when considering how to propagate the updates from one instance to the other, in a state-based approach, that does not take into account the process that has led to the current state (and to the matrix values on each side), one faces an ambiguity: both the column $2Y3$, in $[I_S/[Temp.S]]$, and $2X3$ in $[I_{S'}/[Temp.S']]$, are equal to zero. Yet, in the opposite instance, those columns are not empty: thus, when trying to restore the equality of I_S and $I_{S'}$, should non-null, or null values prevail? Should, after the synchronization of the instances, the two columns $2Y3$ and $2X3$ be both null, both non null, or one of each... ? It appears that this situation is much clearer from an operation-based point of view, which provides a very different interpretation to the nullity of the columns $2Y3$ and $2X3$: no edge labelled Y has been defined in I_S , while an edge labelled X has been *deleted* in $I_{S'}$. The deletion of the edge X should thus be propagated to I_S – and the additions of the edges a and X that have been performed on I_S , to $I_{S'}$.

It thus appears that, in an operation-based approach, propagating an update operated on one side to the other side cannot be done by copying the value of the modified inner matrices of one updated instance into the other instance. Indeed, by doing so, propagating the updates operated on I_S (for instance) first onto $I_{S'}$ by inserting the new values of the modified matrices of $[I_S/[Temp.S]]$ into $[I_{S'}/[Temp.S']]$ would erase some updates done, in the meantime, on $I_{S'}$: the deletion of the edge labelled X between v_2 and v_3 in $I_{S'}$, in particular.

Yet, what makes this last naïve strategy of update propagation impossible – and enforces to have recourse to an operation-based strategy – is the fact that each instance of a pair of instances connected together by a bidirectional transformation have been considered to be updatable simultaneously. In a turn-based model for updates, those situations do not happen: an editor A can only update an instance I_A after the active editor B has committed his changes. Indeed, this model of interactive work is not realistic. We adopt it nonetheless to avoid the burden of having, in this first attempt at defining bidirectional transformations for eAG, to take conflicts into account. Thus the following temporal model for eAG updates:

Definition 10.2 : temporal model for eAG updates. Be $S - I_S$ and $S' - I_{S'}$ two schema-instance pairs related together by means of a pair of SeAG and eAG transformations.

It will be considered that the propagation of an update operated on one of the instances to the other instance is instantaneous: the propagation takes no time and is done as soon as the update is done, regardless of the activity on the other instance.

It will also be considered that the definition of an update onto an instance is instantaneous.

The combination of the two above assumptions is equivalent to saying that the instances are considered to be updatable sequentially.

10.3 Derivation from a *Mod* Modification

10.3.1 eAG *Mod* Behaviour

We have defined our general strategy for the propagation of a SeAG transformation to the domain of the instances: with the nomenclature introduced in Paragraph 10.1, from each modification producing a cast G' out of another cast G , a modification shall produce a graph H' , expressible on the template of G' , out of H .

Thus, let us sum up briefly what a SeAG *Mod* modification consists in, and how it shall be interpreted in the eAG domain.

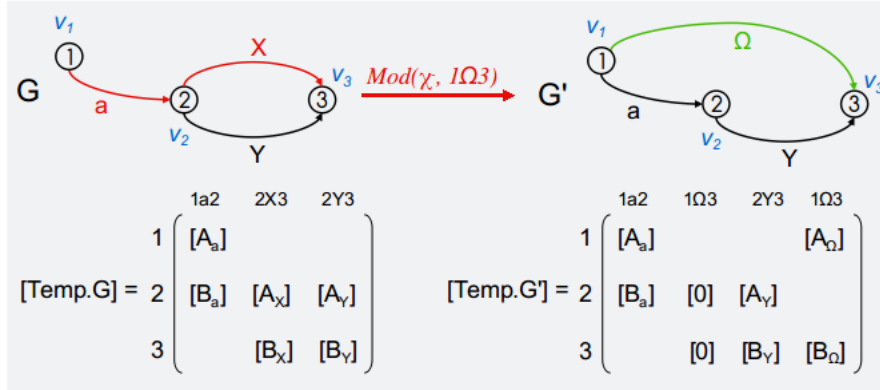
1. A given rooted, single-leafed and connected subgraph of G , made out of one or more root-to-leaf paths that do not intersect, is identified: it is the schematic cell χ^- .

The graph H , represented on $[Temp.G]$, may contain some linear annotation paths that instantiate, or contain instantiations of, some of the root-to-leaf paths of χ^- .

2. Defining the SeAG modification *Mod* with χ^- as a first argument means that, in the image of G , the root-to-leaf paths of χ^- shall not be instantiable anymore. Preventing those to be instantiable is not, in general, done by deleting all the edges of χ^- from G : some of these edges may be involved in paths that are not root-to-leaf paths of χ^- . By analyzing the context of each edge of χ^- , *Mod* attempts to preserve such edges, and to delete the edges that participate to root-to-leaf paths of χ^- only – if possible. In any case, some of the edges of χ^- shall be deleted from G to G' , in order for the root-to-leaf paths of χ^- not to be present in G' . In other words, the inner matrices of $[G/[Temp.G]]^\infty$ that represent those edges are enforced to equal zero in $[Temp.G']^\infty$.

Because H' has to be expressible on $[Temp.G']$, it means that the matrices representing the instances of the deleted edges of G will also be set to zero in $[H'/[Temp.G']]^\infty$. But in the instance domain, this is not a sufficient deletion, in general. Indeed, the purpose of the *Mod* modification on schemas was to suppress the root-to-leaf paths of χ^- from G : in the domain of the instances, this means that the instances of the root-to-leaf paths of χ^- shall be erased – and not only the instances of the edges of χ^- that are deleted in G .

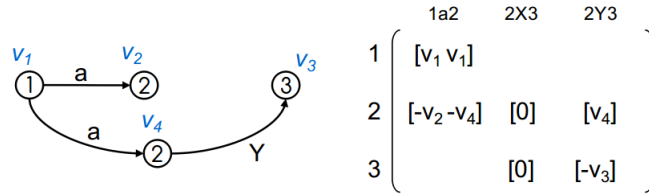
To make this crucial point clearer, let us take an example. Be the following cast G , modified into G' by the substitution of the sequence aX by Ω :



Be the graph H corresponding to G :

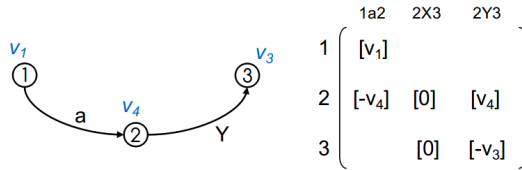


Since the edge X does not belong to $[Temp.G']$, the matrix values corresponding to the instances of that edge in $[H/[Temp.G]]$ shall be turned to zero. But simply doing so will give the following graph:



Instead, all the elements of the matrices $[A_a]$ and $[B_a]$ in $[H/[Temp.G]]$, describing an edge that participates in, and only in, an instance of a root-to-leaf path of χ^- , shall be deleted also. Here; the edge labelled a between v_1 and v_2 was followed, in H , only by an edge labelled X : as such, it should not remain in H' .

Hence the following deletion:



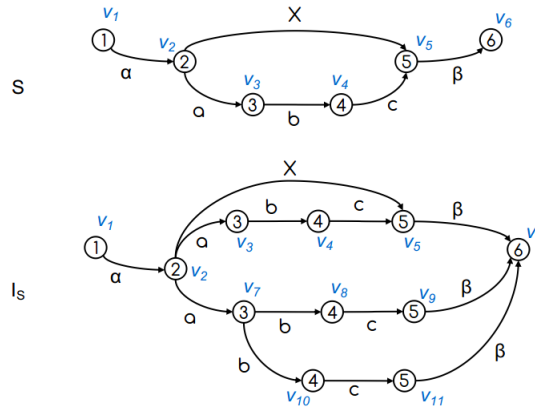
- The cast G' is then obtained by inserting, between the node that was the root of χ^- and the node that was the leaf of χ^- , a substitution cell χ^+ . The propagation of this insertion to the instance domain goes as follows: between all the pairs of nodes that were the corresponding root and leaf of an instance of a root-to-leaf path of χ^+ , an instance of the different root-to-leaf paths of χ^+ must be created. Propagating Mod to the instances thus demands to establish the pairs of root/leaf of the instances of χ^- , and for each, to keep track of the correspondence of this pair with the corresponding pair of root/leaf of an instance of χ^+ . This pair-to-pair correspondence, if it can be read in both directions, can be used to synchronize the two instances: any modification of a path between one pair shall be propagated to a path between the corresponding pair on the other side.

We now present formally how to determine this pair-to-pair correspondence, and how to use it as the basis for a symmetric, graph bidirectional Mod transformation.

10.3.2 Bidirectional eAG Mod Modification Derived from a SeAG Mod Operation

Importantly, the following strategy is defined in a restricted case, that is, in case of independent schematic cells, containing one paths only. Further work will be needed to embrace the general case¹.

Let us consider the following schema-instance pair $S - I_S$ below.



¹Some of the following algorithms have already been extended to the general case: they can be consulted in the Appendix 13 (in French), where they are inserted for the record.

The finite template of S , where $[A_x]$ matrices represent positive matrix variables and $[A_x]$ negative ones, is the following:

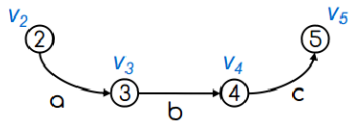
$$[S/[Temp.S]] = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \alpha \\ a \\ b \\ c \\ X \\ \beta \end{array} \begin{bmatrix} [A_{1,1\alpha 2}] & - & - & - & - & - \\ [B_{2,1\alpha 2}] & [A_{2,2a3}] & - & - & [A_{2,2X5}] & - \\ - & [B_{3,2a3}] & [A_{3,3b4}] & - & - & - \\ - & - & [B_{4,3b4}] & [A_{4,4c5}] & - & - \\ - & - & - & [B_{5,4c5}] & [B_{5,2X5}] & [A_{5,5\beta 6}] \\ - & - & - & - & - & [B_{6,5\beta 6}] \end{bmatrix}$$

The expression of S and I_S is done by attributing a certain matrix value to each matrix variable $[A_x]$ and $[B_x]$:

$$S : \left(\begin{array}{ccc} [A_{1,1\alpha 2}] = [v_1] & [A_{2,2a3}] = [v_2] & [A_{3,3b4}] = [v_3] \\ [B_{2,1\alpha 2}] = -[v_2] & [B_{3,2a3}] = -[v_3] & [B_{4,3b4}] = -[v_4] \\ & [A_{4,4c5}] = [v_4] & [A_{2,2X5}] = [v_2] & [A_{5,5\beta 6}] = [v_5] \\ & [B_{5,4c5}] = -[v_5] & [B_{5,2X5}] = -[v_5] & [B_{6,5\beta 6}] = -[v_6] \end{array} \right)$$

$$I_S : \left(\begin{array}{ccc} [A_{1,1\alpha 2}] = [v_1] & [A_{2,2a3}] = [v_2 \ v_2] & [A_{3,3b4}] = [v_3 \ v_7 \ v_7] \\ [B_{2,1\alpha 2}] = -[v_2] & [B_{3,2a3}] = -[v_3 \ v_7] & [B_{4,3b4}] = -[v_4 \ v_8 \ v_{10}] \\ & [A_{4,4c5}] = [v_4 \ v_8 \ v_{10}] & [A_{2,2X5}] = [v_2] & [A_{5,5\beta 6}] = [v_5 \ v_9 \ v_{11}] \\ & [B_{5,4c5}] = -[v_5 \ v_9 \ v_{11}] & [B_{5,2X5}] = -[v_5] & [B_{6,5\beta 6}] = -[v_6 \ v_6 \ v_6] \end{array} \right)$$

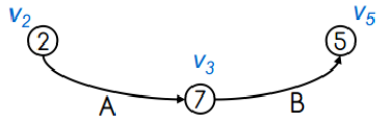
Be then χ^- the following schematic cell and its template-based representation:



$$[Temp.\chi^-] = \begin{array}{c} 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} a \\ b \\ c \end{array} \begin{bmatrix} [A_{2,2a3}] & - & - \\ [B_{3,2a3}] & [A_{3,3b4}] & - \\ - & [B_{4,3b4}] & [A_{4,4c5}] \\ - & - & [B_{5,4c5}] \end{bmatrix}$$

$$\text{with } \chi^- : \left(\begin{array}{ccc} [A_{2,2a3}] = [v_2] & [A_{3,3b4}] = [v_3] & [A_{4,4c5}] = [v_4] \\ [B_{3,2a3}] = -[v_3] & [B_{4,3b4}] = -[v_4] & [B_{5,4c5}] = -[v_5] \end{array} \right)$$

Be the following substitution cell χ^+ :



$$[Temp.\chi^+] = \begin{array}{c} 2 \\ 5 \\ 7 \end{array} \begin{array}{c} A \\ B \end{array} \begin{bmatrix} A & B \\ [A_{2,2A7}] & - \\ - & [B_{7,7B5}] \\ [B_{7,2A7}] & [A_{7,7B5}] \end{bmatrix}$$

$$\text{with } \chi^+ : \left(\begin{array}{cc} [A_{2,2A7}] = [v_2] & [A_{7,7B5}] = [v_7] \\ [B_{7,2A7}] = -[v_7] & [B_{5,7B5}] = -[v_5] \end{array} \right)$$

Let us see how the *Mod* operation defined by the above arguments can be derived into an eAG modification.

First, χ^- is independent, which means that:

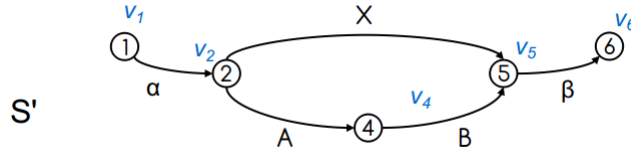
$$[Mod(\chi^-, \chi^+).S/[Temp.Mod(\chi^-, \chi^+).S]]^\infty = [S/[Temp.S]]^\infty - [\chi^-/[Temp.\chi^-]]^\infty + [\chi^+[Temp.\chi^+]]^\infty$$

Thus, by denoting S' the cast defined by $Mod(\chi^-, \chi^+).S$:

$$[Temp.S'] = \begin{matrix} & \alpha & A & B & X & \beta \\ \begin{matrix} 1 \\ 2 \\ 5 \\ 6 \\ 7 \end{matrix} & \begin{bmatrix} [A_{1,A\alpha 2}] & - & - & - & - \\ [B_{2,A\alpha 2}] & [A_{2,2A7}] & - & [A_{2,2X5}] & - \\ - & - & [B_{5,3B5}] & [B_{5,2X5}] & [A_{5,5\beta 6}] \\ - & - & - & - & [B_{6,5\beta 6}] \\ - & [B_{7,2A7}] & [A_{7,3B5}] & - & - \end{bmatrix} \end{matrix}$$

with $S' : \left(\begin{matrix} [A_{11}] = [v_1] & [A_{22}] = [v_2] & [A_{73}] = [v_7] & [A_{24}] = [v_2] & [A_{55}] = [v_5] \\ [B_{21}] = -[v_2] & [B_{72}] = -[v_7] & [B_{53}] = -[v_5] & [B_{54}] = -[v_5] & [B_{65}] = -[v_6] \end{matrix} \right)$

This represents the following schema:



Let us now derive from $Mod(\chi^-, \chi^+)$ a modification translating I_S into an eAG validated by S' . As stated above, the target is to replace the edge sequences, in I_S , that instantiate a root-to-leaf path of χ^- by an instance of χ^+ .

It is worth noting here that $[I_S/[Temp.\chi^-]]$ contains the description of all the edges, in I_S that instantiate an edge from χ^- . Yet it is among those edges that the edges belonging to a sequence instantiating a complete root-to-leaf path of χ^- have to be looked for. As we have seen in Paragraph 10.3.1, the information that is needed, for establishing the bidirectional modification in the instance domain, is the set of root/leaf pairs of such complete sequences: to establish this set, the graph represented by $[I_S/[Temp.\chi^-]]$ has to be visited, starting from the nodes that can possibly be roots for such complete root-to-leaf sequences, down to the possible leaves. We define such a visiting method on the matrix representation of the graph below:

1. χ^- has one and only one root, characterized by the fact that it corresponds to the only block-line in $[Temp.\chi^-]$ where there is no negative matrix variable. Here, the type of the root of χ^- is thus 2.

As a consequence, only the nodes whose type is 2 may be the root for an instance of a root-to-leaf path from χ^- in I_S . The visit of the matrix shall then start by the inner matrices that belong to the block-line of type 2: $[A_{2,2a3}] = [v_1 \ v_2]$ here.

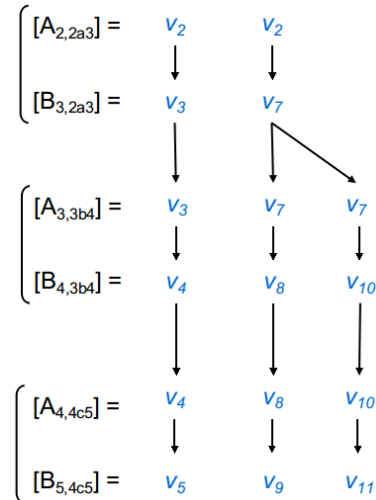
2. It is then possible to know the following node of each node in $[A_{2,2a3}] = [v_1 \ v_2]$. It suffices to look for the only inner matrix that is not equal to zero on the same block-column as $[A_{2,2a3}]$, that is $[B_{3,2a3}] = -[v_3 \ v_7]$ here. Then we know that

the only sequences of edges, in I_S , that may instantiate a root-to-leaf path of χ^- , are the sequences starting by an edge labelled a between v_2 and v_3 on the one hand, and between v_2 and v_7 on the other hand.

3. Before continuing the visit, one must check whether the type of the nodes v_3 and v_7 is equal to the type of the leaf of χ^- – which would indicate that the visit is finished. It is not the case since the block-line corresponding to $type(v_3) = type(v_7) = 3$ in $[I_S/[Temp.\chi^-]]$ does not contain negative matrices only.
4. The type of the deepest nodes reached so far is not the type of the leaf of χ^- : this means that the nodes v_3 and v_7 , may appear in at least one positive matrix on the same block-line as the previously reached, negative matrix $[B_{3,2a3}]$. There is only one such positive matrix: $[A_{3,3b4}] = [v_3 \ v_7 \ v_7]$. Both v_3 and v_7 are thus in turn the summit of an edge instantiating an edge of χ^- . Additionally, we can note that v_7 is actually the summit of two such edges.
5. The same process is applied recursively, by looking for the ends of the edges whose roots have been discovered, and then checking if those ends are summits of further edges instantiating an edge of χ^- – until reaching nodes whose type equals the type of the leaf of χ^- . Handling the cycles here is quite easy: the only cycles in the eAG domain involve links. Thus, a record of the edges whose label contains `:LinkTo` has to be kept, not to visit their end twice.

By doing as indicated above, one finds that the next matrix after $[A_{3,3b4}]$ is $[B_{4,3b4}] = -[v_4 \ v_8 \ v_{10}]$. Those nodes then occur, on the same block-line, in $[A_{4,4c5}] = [v_4 \ v_8 \ v_{10}]$. They then lead to the nodes $[B_{5,4c5}] = [v_5 \ v_9 \ v_{11}]$ by means of another edge instantiating an edge of χ^- . There is no positive matrix on the block-line 5, so the algorithm finishes here.

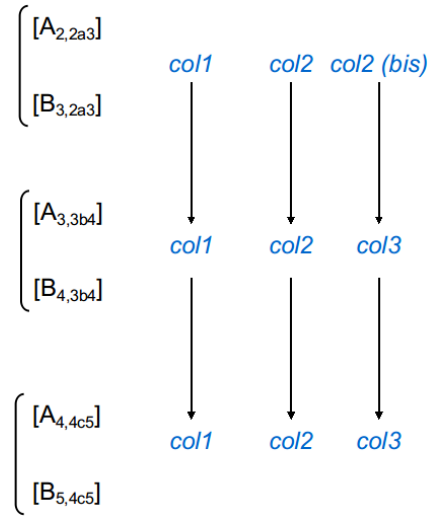
Starting from the candidate nodes for being the roots of the instance of a root-to-leaf path of χ^- , the above visiting procedure provides the sequence of nodes that follow those candidate roots. We can represent this information as follows:



This representation of the paths that have been identified in $[I_S/[Temp.\chi^-]]$ illustrates the fact that the matrices $([A_{ij}], [B_{kjl}])$ form pairs within which the node designated by the n^{th} column from $[A_{ij}]$ belongs to the same path as the node designated by the n^{th} column of $[B_{kj}]$ (this is natural since this pair of nodes are the summit and end of an edge of $[I_S/[Temp.\chi^-]]$).

More interestingly, this representation also shows that a node represented by the k^{th} column of a given $[B_{kj}]$, and belonging to an instance of a root-to-leaf path of χ^- , will occur in another matrix $[A_{kl}]$ from the same block-line of $[I_S/[Temp.\chi^-]]$; more precisely, several columns from the matrix $[A_{kl}]$ may represent the same node as the n^{th} column of a given $[B_{kj}]$, in the case where that node is the summit of several edges instantiating an edge from χ^- .

Consequently, one way to represent the different paths instantiating a root-to-leaf path of χ^- is by stating to which column(s) of a pair $([A_{ij}], [B_{kj}])$ align with which column(s) of the following pair $([A_{kl}], [B_{ml}])$, as follows:



Based on the information given on the right side of the above representation, we know that the node represented by the first column of the matrix $[A_{2,2a3}]$ is the root of a path whose leaf is represented by the first column of the matrix $[B_{5,4c5}]$, while the second node of $[A_{2,2a3}]$ is the root of two paths: one ending on the second node of $[B_{5,4c5}]$, and the other ending on the third.

The conclusion we draw from the above considerations is that, when χ^- contains one single path, it is possible to represent the output from the matrix visit by a sequence of pairs $([A_{ij}], [B_{kjl}])$ and a series of (special kinds of) permutations describing how the columns of this pair relate with the columns of the next. By composing all the permutations, the pairs of nodes that work as the root and the leaf of the same instance of a path of χ^- will be given.

In the present case, two permutations can be defined: σ_1 between the columns of the matrices defining the edges characterized by the triple $(2, a, 3)$ and the columns of those defining the edges $(3, b, 4)$; σ_2 , between the columns of $(3, b, 4)$ and $(4, c, 5)$.

In the current context where χ^- contains a single path, we define those permutations as follows:

1. Be two matrices $[B_{ij}]$ and $[A_{il}]$ whose columns are to be aligned. Be L_B the list of the node identifiers in their order of appearance, from left to right, in $[B_{ij}]$, and L_A the corresponding list for $[A_{il}]$. For all index k over the elements of L_B , if $\exists K = \{k_1, k_2 \dots k_N\}$ so that $\forall k_i, (L_A)_{k_i} = (L_B)_k$, then we define a permutation $\sigma(k)$ of \mathbb{N}^2 by means of a matrix whose elements are, on the first line, several occurrences of k together with an additional flag value in $\{n \in \mathbb{N}, n > 1\}$, and on the second line, the k_i values:

$$\sigma(k) = \begin{pmatrix} k & k^{(2)} & \dots & k^{(2)} \\ k_1 & k_2 & \dots & k_N \end{pmatrix}$$

The complete permutation σ between the columns of $[B_{ij}]$ and $[A_{il}]$ is the concatenation of all the $\sigma(k)$, for k varying over the set of indexes over the list L_B .

2. Symmetrically, if the second line of the σ contains several occurrences of the same value, those will be distinguished by means of flags as indicated above.

This way, σ will indeed be a permutation of \mathbb{N}^2 , each flagged index being a pair of naturals, and a non-flagged one corresponding to the case where the flag shall equal 1.

In our example :

1. Between $[B_{3,2a3}] = -[v_3 \ v_7]$ and $[A_{3,3b4}] = [v_3 \ v_7 \ v_7]$: the node designated by the first column of $[B_{3,2a3}]$ only matches the first column of $A_{3,3b4}$; the node designated by the second column of $[B_{3,2a3}]$ matches both the second and the third columns of $A_{3,3b4}$;

$$\sigma_1 = \begin{pmatrix} 1 & 2 & 2^{(2)} \\ 1 & 2 & 3 \end{pmatrix}$$

2. Between $[B_{4,3b4}] = -[v_4 \ v_8 \ v_{10}]$ and $A_{4,4c5} = -[B_{4,3b4}]$: the permutation is the identity:

$$\sigma_2 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$$

In order to determine the pairs of nodes working as the root and its corresponding leaf of an instance of χ^- , we calculate the composition of those two permutations:

$$\sigma = \sigma_2 \circ \sigma_1 = \begin{pmatrix} 1 & 2 & 2^{(2)} \\ 1 & 2 & 3 \end{pmatrix}$$

which indicates that there are three instances of χ^- in I_S :

- the first between the node designated by the first column of $[A_{2,2a3}]$ and the node designated by the first column of $[B_{5,4c5}]$, that is between v_2 and v_5 ;

- the second between the node designated by the second column of $[A_{2,2a3}]$ and the node designated by the second column of $[B_{5,4c5}]$, that is between v_2 and v_9 ;
- the last between the node designated by the second column of $[A_{2,2a3}]$ also, and the node designated by the third column of $[B_{5,4c5}]$, that is between v_2 and v_{11} .

To fulfil the propagation, we proceed as follows:

All the paths of χ^+ will have to be instantiated between each root-leaf pair identified just above. This is done by creating the needed nodes and edges:

1. Be i_r the index of the type value of the root of χ^+ . Then for j so that there is a positive matrix $[A_{i_r,j}]$ in the template of χ^+ , $[A_{i_r,j}]$ is given as a value the ordered list of the roots of the instances of χ^- found above – a multiple root being repeated adequately in $[A_{i_r,j}]$.
Here, this gives: $[A_{2,2A7}] = [v_2 \ v_2 \ v_2]$.
2. The same process shall be done for the leaves: Be i_l the index of the type value of the leaf of χ^+ . Then for j so that there is a negative matrix $[B_{i_l,j}]$ in the template of χ^+ , $[B_{i_l,j}]$ is given as a value the ordered list of the leaves of the instances of χ^- found above – a multiple root being repeated adequately in $[B_{i_l,j}]$.
This gives $[B_{7,7B5}] = [v_5 \ v_9 \ v_{11}]$.
3. Then, all the other matrices of $[Temp.\chi^+]$ have to be given a value so that it verifies the following rules:

$$\begin{aligned} \alpha. \forall i, i', j, size_W. [A_{ij}] &= size_W. [B_{i'j}] \\ \beta. \forall i, l, x, y \mid [G/[Temp.G]_{il}]_{xy} &= -1, \text{ so that } l \text{ is not the index of the} \\ &\text{type value of the leaf of } \chi^+ : \\ \exists l', y' \mid [G/[Temp.G]_{il'}]_{xy'} &= 1. \end{aligned}$$

To achieve that, we propose the following procedure:

- (a) For the value j so that $[A_{i_r,j}] \neq [0]$, find the value i' so that $[B_{i'j}]$ is defined. If $i' \neq i_l$, then fill $[B_{i'j}]$ with a negative list of newly created node identifiers, so that $size_W. [A_{i_r,j}] = size_W. [B_{i'j}]$.
Here, for instance: $[B_{7,5A7}] = -[v_{12} \ v_{13} \ v_{14}]$.
- (b) Then find j' so that $[A'_{i'j'}]$ is defined, and give $[A'_{i'j'}]$ the value $-[B_{i'j}]$. This ensures that the above rule β is respected.
Here: $[A_{7,7B5}] = [v_{12} \ v_{13} \ v_{14}]$.

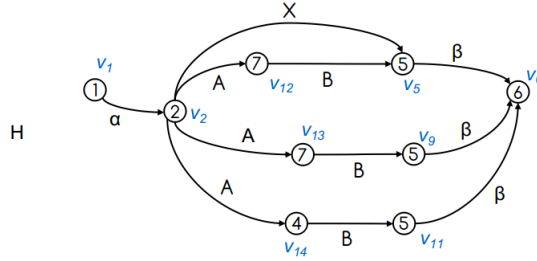
Repeat steps a. and b. replacing $[A_{i_r,j}]$ by $[A'_{i'j'}]$ above.

Here, the procedure stops without repeating it, since 7 is the index of the type value of the leaf of χ^+ .

In summary, the whole forward propagation of $Mod(\chi^-, \chi^+)$ to I_S yields the following representation on the template of S' :

$$\left(\begin{array}{ccc} [A_{1,1\alpha 2}] = [v_1] & [A_{2,2A7}] = [v_2 \ v_2 \ v_2] & [A_{7,7B5}] = [v_{12} \ v_{13} \ v_{14}] \\ [B_{2,1\alpha 2}] = -[v_2] & [B_{7,5A7}] = -[v_{12} \ v_{13} \ v_{14}] & [B_{5,7B5}] = [v_5 \ v_9 \ v_{11}] \\ & & [A_{2,2X5}] = [v_2] & [A_{5,5\beta 6}] = [v_5 \ v_9 \ v_{11}] \\ & & [B_{5,2X5}] = -[v_5] & [B_{6,5\beta 6}] = -[v_6 \ v_6 \ v_6] \end{array} \right)$$

which represents the following graph $I_{S'}$:



We leave the question of how to affect a reference value to the created nodes open until Paragraph 10.6.

In the end, since it is represented over the template of S' , this graph, which also happens to be a well-formed eAG, is validated by S' . We can also consider that it captures as much information from I_S as possible, and translates each path abc into a path AB as expected.

Nota. We can also note that the compositional properties given in Paragraph 10.1 are guaranteed by construction: Property number 1. page 254 corresponds exactly to the rule α above, and Property number 2., which requires an identifier not to appear on two block-lines of the template-based expression of the image graph, is also guaranteed by the fact that new identifiers are created whenever a new block-line is considered in the above procedure.

Now let us illustrate how this forward initialization of $I_{S'}$ based on I_S can be interpreted bidirectionally, and serve as a synchronization mechanism between the two instances. To do that, we first need to define how to operate a permutation on a matrix:

Definition 10.3 : Permutation on a matrix. Be $[M]$ a matrix that can be represented as a list $\pm[m_1 \ m_2 \ \dots \ m_N]$, where each m_i is a column vector containing one and only one non-null coefficient, that is equal to 1..

Be σ a permutation of \mathbb{N}^2 so that for all $i \in [1; N]$, and for all $k \geq 1$, $\exists!(j, l) \in \mathbb{N}^2$ so that $\sigma(i^{(k)}) = j^{(l)}$, and so that:

$$\begin{aligned} \forall i, i' \in [1; N], \forall k, k' \in \mathbb{N}, \sigma(i^{(k)}) = j^{(l)} = \sigma(i'^{(k')}) \\ \Leftrightarrow i = i' \wedge k = k'. \end{aligned}$$

Then $\sigma.[M]$ is the matrix defined by:

1. For i, j, k so that $\sigma(i^{(k)}) = j^{(1)}$, then: $[\sigma.[M]]_j = [M]_i$
2. For $i, j, k, l \neq 1$ so that $\sigma(i^{(k)}) = j^{(l)}$, then, with '+' denoting the "sum" operator defined over $\{-1, 0, 1\}$ in Table 9.1 page 223 and $[\sigma.[M]]*_j$ the value of the coefficient before operating the sum operator:

$$[\sigma.[M]]_j = [\sigma.[M]]*_j + [M]_i$$

Example. Let us consider $[B_{3,2a3}] = -[v_3 \ v_7]$ and the permutation σ_1 defined above by the following matrix:

$$\sigma_1 = \begin{pmatrix} 1 & 2 & 2^{(2)} \\ 1 & 2 & 3 \end{pmatrix}$$

There are three triples (i, j, k) so that $\sigma(i^{(k)}) = j$: $(1, 1, 1)$ $(2, 2, 1)$ and $(2, 3, 2)$. Thus, $\sigma.[B_{3,2a3}] = -[v_3 \ v_7 \ v_7]$.

Nota. We can notice the in the above example, with σ_1 defined as the permutation between the columns of $[B_{3,2a3}]$ and $[A_{3,3b4}]$, we have:

$$\sigma.[B_{3,2a3}] = -[A_{3,3b4}]$$

This property is actually a general result in the case of schematic cells containing one path only:

Property 10.1 Be σ the permutation between the lines of two matrices $[B_{ij}]$ and $[A_{ik}]$. Then:

$$\sigma.[B_{ij}] = -[A_{ik}]$$

and naturally:

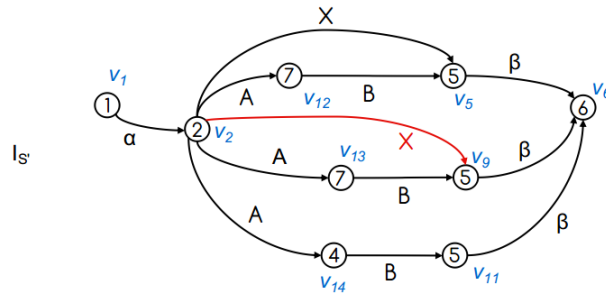
$$\sigma^{-1}.[A_{ik}] = [B_{ij}]$$

Nota. In a schematic cell containing one path only, there is at most one pair of positive and negative matrices per block-line – hence the above result. The extension to the general case is treated in the next paragraph.

Let us now consider how those properties enable to bidirectionalize the transformation that gave $I_{S'}$ out of I_S .

1. First, if the properties of a node that is common between I_S and $I_{S'}$ is modified in one of the instances (i.e. its reference value is changed), the same change must be operated on the other instance. If an edge belonging to the instances of an edge from $S \cap S'$ is deleted or added in one of the instances, it must be deleted or inserted in the other as well. This elementary update propagations can be done, in the temporal model we chose, by copying the matrix values from the expression of one instance to the other.

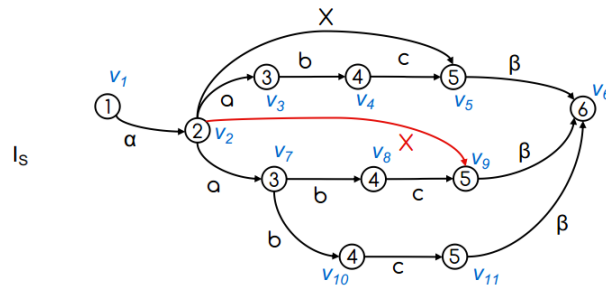
For instance, let us make the assumption that an edge X has been inserted in the instance $I_{S'}$ defined above, as follows:



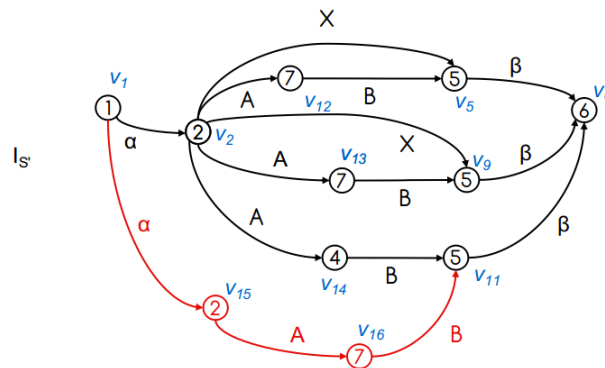
In terms of the value of the matrix variables describing $I_{S'}$, this update is defined as follows:

- $[A_{2,2X5}] = [v_2] \longrightarrow [A_{2,2X5}] = [v_2 \ v_2]$;
- $[B_{5,2X5}] = [v_5] \longrightarrow [B_{5,2X5}] = [v_5 \ v_9]$.

If those two new matrix variable values are copied in the list of matrix values $\{I_S/[Temp.S]\}$ describing I_S , we have:



2. Let us now consider the following update: an edge sequence αAB is inserted in $I_{S'}$ between the nodes v_1 and v_{11} :



The matrix description of this update is:

- $[A_{1,1\alpha 2}] = [v_1] \longrightarrow [A_{1,1\alpha 2}] = [v_1 \ v_1]$
 $[B_{2,1\alpha 2}] = -[v_2] \longrightarrow [B_{2,1\alpha 2}] = -[v_2 \ v_{15}] ;$
- $[A_{2,2A7}] = [v_2 \ v_2 \ v_2] \longrightarrow [A_{2,2A7}] = [v_2 \ v_2 \ v_2 \ v_{15}]$
 $[B_{7,2A7}] = -[v_{12} \ v_{13} \ v_{14}] \longrightarrow [B_{7,2A7}] = -[v_{12} \ v_{13} \ v_{14} \ v_{16}] ;$
- $[A_{7,7B5}] = [v_{12} \ v_{13} \ v_{14}] \longrightarrow [A_{7,7B5}] = [v_{12} \ v_{13} \ v_{14} \ v_{16}]$
 $[B_{5,7B5}] = [v_5 \ v_9 \ v_{11}] \longrightarrow [B_{5,7B5}] = [v_5 \ v_9 \ v_{11} \ v_{11}].$

Propagating this update can be done in three steps:

- (a) First, the value of the common matrices between $\{I_{S'}/[Temp.I_{S'}]\}$ and $\{I_S/[Temp.I_S]\}$ are copied from the updated instance to the other: for example, in $\{I_S/[Temp.I_S]\}$:

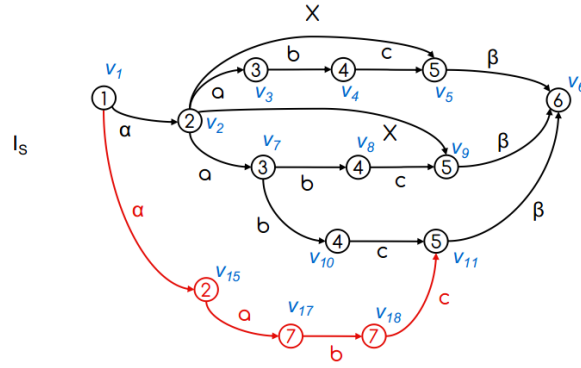
$$[A_{1,1\alpha 2}] = [v_1] \longrightarrow [A_{1,1\alpha 2}] = [v_1 \ v_1]$$

$$[B_{2,1\alpha 2}] = -[v_2] \longrightarrow [B_{2,1\alpha 2}] = -[v_2 \ v_{15}]$$

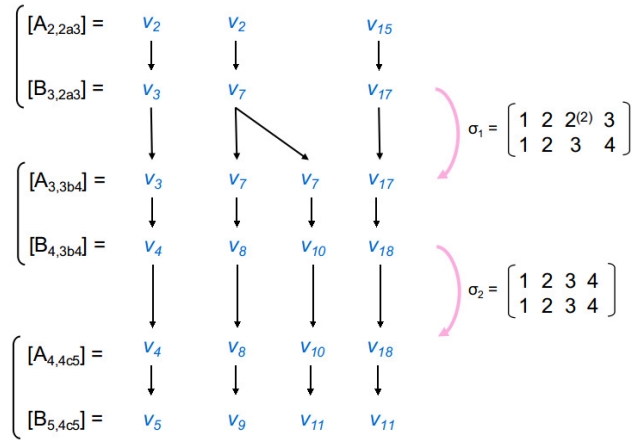
- (b) Additionally, it has to be considered that the update on $I_{S'}$ also impacts the value of the matrices from $[I_{S'}/[Temp/\chi^+]]$. The first thing to check is whether this update affects the pairs of roots and leaves of the instances of χ^+ in $I_{S'}$. The procedure to obtain the list of such pairs is the one described above. It provides the following pairs: $\{(v_2, v_5); (v_2, v_9); (v_2, v_{11}); (v_{15}, v_{11})\}$. Had the update not affected the list of root-leaf pairs, it would not have to be propagated² Yet, because this list differs from the previous one, the update operated on the matrix values from $\{I_{S'}/[Temp.I_{S'}]$ will have to be propagated to the other instance: a new root-leaf pair has been defined in $I_{S'}$, namely (v_{15}, v_{11}) . Thus, because the nodes v_{15} and v_{11} have to be part of the two instances, an instance of χ^- has to be defined between v_{15} and v_{11} in I_S . The procedure for creating this instance is the same as the one defined above for initializing the instances of χ^+ in $I_{S'}$. Hence the following new value for I_S :

²For instance, suppose that the graph $I_{S'}$ below, left, has been replaced by the new graph $I_{S'}$ (on the right). The pairs of nodes playing the role of a root and leaf for an instance of a path of χ^+ is left untouched by this update: it needs not be transposed onto the other instance then.

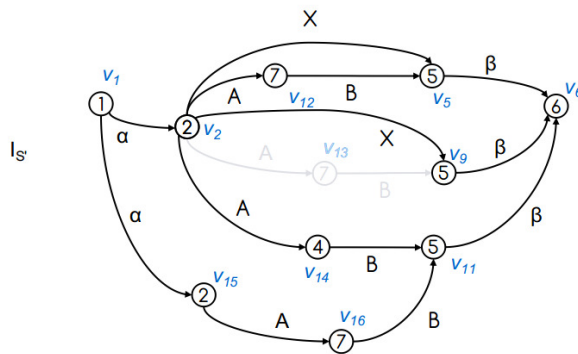




Nota Bene. The permutations between the matrix pairs in $[I_S/[Temp.S]]$, that had been calculated for the old version of I_S , can be easily updated. They become:



3. To finish with, let us make the assumption now that a path, instantiating χ^+ , is deleted in I_S :



The new matrix values in $\{I_{S'}/[Temp/S']\}$ are then:

- $[A_{2,2A7}] = [v_2 \ v_2 \ v_2 \ v_{15}] \longrightarrow [A_{2,2A7}] = [v_2 \ 0 \ v_2 \ v_{15}]$
 $[B_{7,2A7}] = -[v_{12} \ v_{13} \ v_{14} \ v_{16}] \longrightarrow [B_{7,2A7}] = -[v_{12} \ 0 \ v_{14} \ v_{16}] ;$
- $[A_{7,7B5}] = [v_{12} \ v_{13} \ v_{14} \ v_{16}] \longrightarrow [A_{7,7B5}] = [v_{12} \ 0 \ v_{14} \ v_{16}]$
 $[B_{5,7B5}] = [v_5 \ v_9 \ v_{11} \ v_{11}] \longrightarrow [B_{5,7B5}] = [v_5 \ 0 \ v_{11} \ v_{11}].$

Propagating a deletion is very different from propagating an insertion: the second requires to create from scratch an instance of the schematic cell in the graph the insertion is propagated to, and this is done by adding a new column, with new identifiers, in each of the non-null matrices from $[I_S/[Temp.\chi]]$. Propagating a deletion demands to intervene in a more subtle way onto the matrices of $[I_S/[Temp.\chi]]$. For instance, as we will see, in the case of the deletion illustrated above, the matrix of $[I_S/[Temp.\chi]]$ describing the roots of the instances of χ^- will remain untouched.

Two additional notions are thus needed for the propagation of a deletion: the deletion matrix and the permutation between the roots of the two instances.

Definition 10.4 : Deletion matrix. Be $[M]$ a $n \times m$ matrix. Be $[M']$ another $n \times m$ matrix, obtained by turning some of the columns of $[M]$ to zero. The deletion matrix $[\Delta_{M,M'}^{suppr.}]$ is the $n \times m$ matrix that describes the shift from $[M]$ to $[M']$. It is defined by the following equality:

$$[M'] = [\Delta_{M,M'}^{suppr.}] \odot [M]$$

where \odot denotes the Hadamard product.

Notation. In the following, we will have to designate the value of a given matrix variable $[M]$ before and after an update. Then $[M]*$ will be used to designate the value before the update, and $[M]$ the updated value.

We will also have to write deletion matrices whose columns contain either only zeros or ones. We will write those matrices in the shape of a vector of ones and zeros (see the example below).

Example. In the example we started above, in $[I_{S'}/[Temp.\chi^+]]$:

$$[A_{2,2A7}] = [v_2 \ v_2 \ v_2 \ v_{15}] \longrightarrow [A_{2,2A7}]_1 = [v_2 \ 0 \ v_2 \ v_{15}],$$

Hence the corresponding deletion matrix:

$$[\Delta_{[A_{2,2A7}]_0, [A_{2,2A7}]_1}^{suppr.}] = [\ 1 \ 0 \ 1 \ 1 \]$$

Definition 10.4: Root-root permutation. Be two schema-instance pairs (S, I_S) and $(S', I_{S'})$, so that there are two schematic cells $\chi^- \subseteq S$ and $\chi^+ \subseteq S'$ verifying $S' = Mod(\chi, \chi^+).S$. Be i the index of the type of the root of χ^- and χ^+ in \mathcal{T} . We make the assumption here that there is a single positive matrix $[A]$ ($[A']$, respectively) on the i^{th} block-line of $[\chi^-/[Temp.\chi^-]]$ (resp. $[\chi^+[Temp.\chi^+]]$).

We define $\sigma_{S,S'}^{root}$ the permutation so that:

$$\sigma_{S,S'}^{root} \cdot [A] = [A']$$

Nota. This permutation always exists in the context of independent cells, since in this case, the only nodes that appear in $[A]$ will be the root for an instance of χ^- – and hence, a root for an instance of χ^+ in $I_{S'}$ – and conversely.

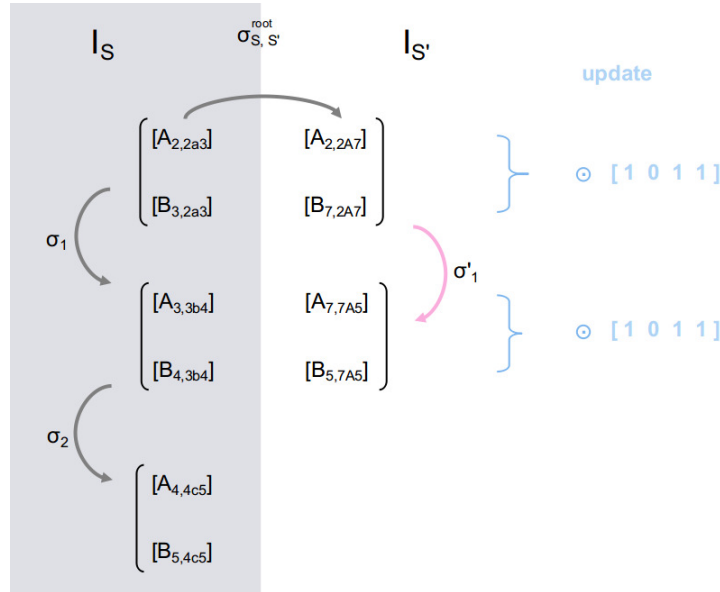
Example. In the running example, we have seen that:

- $[A] = [A_{2,2a3}] = [v_2 \ v_2 \ v_{15}]$
- $[A'] = [A_{2,2A7}] = [v_2 \ v_2 \ v_2 \ v_{15}]$

Thus:

$$\sigma_{S,S'}^{root} = \begin{pmatrix} 1 & 2 & 2^{(2)} & 3 \\ 1 & 2 & 3 & 4 \end{pmatrix}$$

We can now define the way a deletion is propagated from one instance to the other. The situation is the following: we have already defined the permutations that describe the instances of χ^- and χ^+ in terms of the correspondence of the columns of the matrices from $[I_S/[Temp.\chi^-]]$ and $[I_{S'}/[Temp.\chi^+]]$; we have identified the root-root permutation between the roots of the instances of χ^- and χ^+ . An update is performed on $I_{S'}$: it is a deletion of two edges, and can be described by the two deletion matrices below (in blue).



Propagating this deletion to the matrices of $\{I_S/[Temp.S]\}$ will be done by exploiting the numerous permutation matrices, that enable to align the columns of the matrices of each instance with the columns describing the roots of the instances of the cells they contain (σ_1 , σ_2 and σ'_1) or the roots of the two instances ($\sigma_{S,S'}^{root}$): by transitivity, thanks to all those permutations, each column of any matrix of $[I_{S'}/[Temp.\chi^+]]$ can thus be

aligned with any column of any matrix of $[I_S/[Temp.\chi^-]]$, and conversely. The propagation of a deletion thus consists in setting to zero the columns, in $[I_S/[Temp.\chi^-]]$, that align with deleted columns from $[I_{S'}/[Temp.\chi^+]]$:

- The new value of $[A_{2,2a3}]$ is calculated by :

$$\begin{aligned}
 [A_{2,2a3}] &= \sigma_{S,S'}^{root-1} \circ \sigma_1^{-1} \cdot [\Delta_{[A_{7,7B5}]^*, [A_{7,7B5}]}^{suppr.}] \odot [A_{2,2a3}]^* \\
 &= \sigma_{S,S'}^{root-1} \cdot [\Delta_{[A_{2,2A7}]^*, [A_{2,2A7}]}^{suppr.}] \odot [A_{2,2a3}]^* \\
 &= \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 2^{(2)} & 3 \end{pmatrix} \cdot [1 \ 0 \ 1 \ 1] \odot [v_2 \ v_2 \ v_{15}] \\
 &= [1 \ 1 \ 1] \odot [v_2 \ v_2 \ v_2 \ v_{15}] \\
 &= [v_2 \ v_2 \ v_{15}]
 \end{aligned}$$

- Then, the calculation of the new value of $[B_{3,2a3}]$ is:

$$\begin{aligned}
 [B_{3,2a3}] &= \sigma_{S,S'}^{root-1} \cdot [\Delta_{[A_{2,2A7}]^*, [A_{2,2A7}]}^{suppr.}] \odot [B_{3,2a3}]^* \\
 &= [1 \ 1 \ 1] \odot [v_3 \ v_7 \ v_{17}] \\
 &= [v_3 \ v_7 \ v_{17}]
 \end{aligned}$$

- The calculation of the new value of $[A_{3,3b4}]$ is:

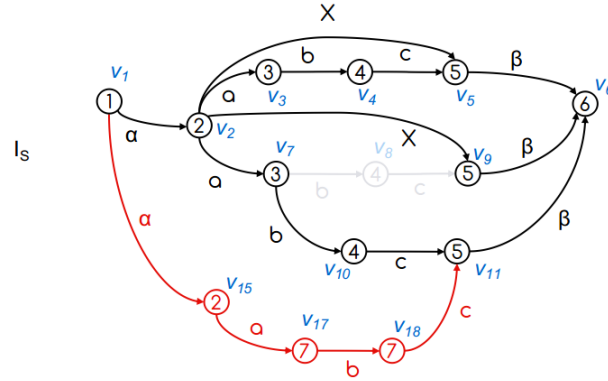
$$\begin{aligned}
 [A_{3,3b4}] &= \sigma_1 \circ \sigma_{S,S'}^{root-1} \circ \sigma_1^{-1} \cdot [\Delta_{[A_{7,7B5}]^*, [A_{7,7B5}]}^{suppr.}] \odot [A_{3,3b4}]^* \\
 &= \begin{pmatrix} 1 & 2 & 2^{(2)} & 3 \\ 1 & 2 & 3 & 4 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 2^{(2)} & 3 \end{pmatrix} \cdot [\Delta_{[A_{7,7B5}]^*, [A_{7,7B5}]}^{suppr.}] \odot [A_{3,3b4}]^* \\
 &= id. [1 \ 0 \ 1 \ 1] \odot [v_3 \ v_7 \ v_7 \ v_{17}] \\
 &= [v_3 \ 0 \ v_7 \ v_{17}]
 \end{aligned}$$

- Etc.

The resulting list of matrix values describing the updated I_S is then:

$$\left(\begin{array}{ccc} [A_{1,1\alpha 2}] = [v_1 \ v_1] & [A_{2,2a3}] = [v_2 \ v_2 \ v_{15}] & [A_{3,3b4}] = [v_3 \ 0 \ v_7 \ v_{17}] \\ [B_{2,1\alpha 2}] = -[v_2 \ v_{15}] & [B_{3,2a3}] = -[v_3 \ v_7 \ v_{17}] & [B_{4,3b4}] = -[v_4 \ 0 \ v_{10} \ v_{18}] \\ & [A_{4,4c5}] = [v_4 \ 0 \ v_{10} \ v_{18}] & [A_{2,2X5}] = [v_2 \ v_2] & [A_{5,5\beta 6}] = [v_5 \ v_9 \ v_{11}] \\ & [B_{5,4c5}] = -[v_5 \ 0 \ v_{11} \ v_{11}] & [B_{5,2X5}] = -[v_5 \ v_9] & [B_{6,5\beta 6}] = -[v_6 \ v_6 \ v_6] \end{array} \right)$$

This represents the following graph, that does reflect all the updates on $I_{S'}$ that have been performed throughout this paragraph:



It has to be noted, in particular, that this method manages the bifurcation between the two instances of χ^- , located on v_7 : the edge $v_2[a]v_7$ has not been deleted, even though it belonged to a path instantiating χ^- that, as a consequence of the deletion in $I_{S'}$, has been impacted by the propagated deletion.

10.4 Derivation from a *Split* Modification

A SeAG modification $Split(i, j, k)$ somehow has an opposite job, in the SeAG field, to a SeAG modification $Unite(j, k, i)$. Of course, $Unite$ and $Split$ cannot be regarded as the inverse one of the other in the field of schemas, in particular because the domain $Unite$ is defined on comprises more than the restricted family of graphs $Split$ produces, in which the degree of the summit and end of the newly created ϵ edges has to be equal to one. Yet, for that reason, $Unite$ restricted to this family of graphs does behave like the inverse of $Split$. Thus, in the eAG field, we will regard the eAG modification derived from $Split(i, j, k)$ as the inverse³ of the eAG modification derived from $Unite(j, k, i)$.

We will thus define the derivation of $Unite$ in the following paragraph only.

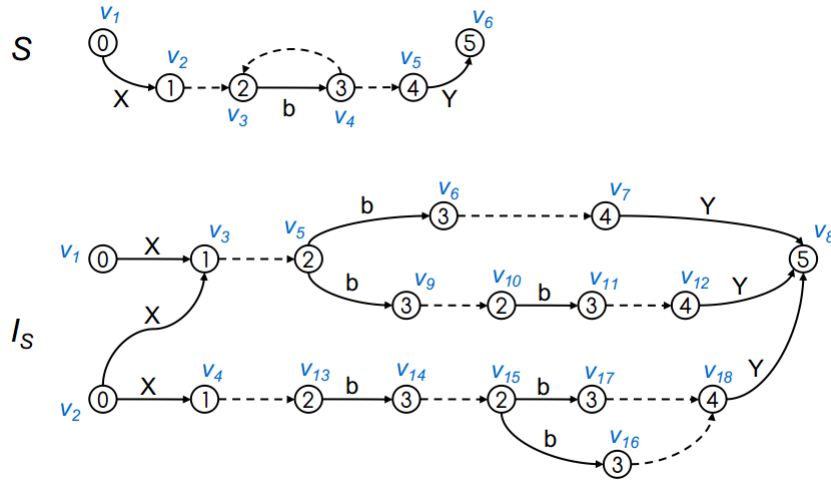
10.5 Derivation from a *Unite* Modification

As the last contribution in this chapter, let us now consider how to derive an eAG modification from a SeAG $Unite(j, k, i)$ modification.

Let us consider the following schema and the graph I_S , expressible on the template of this schema⁴. The different ϵ edges are distinguished by means of subscripts here for the sake of clarity.

³In our symmetric bidirectional transformation context, the inverse of a forward-backward transformation is the corresponding backwards-forward transformation.

⁴Which is not a well-formed eAG in this case.



$$\begin{array}{c}
 [Temp.S] = \\
 \begin{array}{c}
 X \quad \epsilon_1 \quad b \quad \epsilon_2 \quad \epsilon_3 \quad Y \\
 \begin{array}{c}
 0 \left[\begin{array}{c} [A_{0,0X1}] \quad - \quad - \quad - \quad - \quad - \\ [B_{1,0X1}] \quad [A_{1,1\epsilon_1 2}] \quad - \quad - \quad - \quad - \\ - \quad [B_{2,1,1\epsilon_1 2}] \quad [A_{2,2b3}] \quad [B_{2,3\epsilon_2 2}] \quad - \quad - \\ - \quad - \quad [B_{3,2b3}] \quad [A_{3,3\epsilon_2 2}] \quad [A_{3,3\epsilon_3 4}] \quad - \\ - \quad - \quad - \quad - \quad [B_{4,3\epsilon_3 4}] \quad [A_{4,4Y5}] \\ - \quad - \quad - \quad - \quad - \quad [B_{5,4Y5}] \end{array} \right]
 \end{array}
 \end{array}
 \end{array}$$

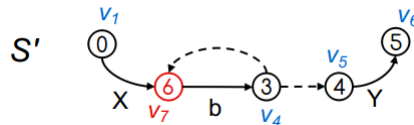
The expression of S over its own template is:

$$\{S/[Temp.S]\} = \begin{pmatrix} [A_{0,0X1}] = [v_1] & [B_{1,0X1}] = -[v_2] \\ [A_{1,1\epsilon_1 2}] = [v_2] & [B_{2,1,1\epsilon_1 2}] = -[v_3] \\ [A_{2,2b3}] = [v_3] & [B_{3,2b3}] = -[v_4] \\ [A_{3,3\epsilon_2 2}] = [v_4] & [B_{2,3\epsilon_2 2}] = -[v_3] \\ [A_{3,3\epsilon_3 4}] = [v_4] & [B_{4,3\epsilon_3 4}] = -[v_5] \\ [A_{4,4Y5}] = [v_5] & [B_{5,4Y5}] = -[v_6] \end{pmatrix}$$

The expression of I_S over its own the template of S is:

$$\{I_S/[Temp.S]\} = \begin{pmatrix} [A_{0,0X1}] = [v_1 \ v_2 \ v_2] & [B_{1,0X1}] = -[v_3 \ v_3 \ v_4] \\ [A_{1,1\epsilon_1 2}] = [v_3 \ v_4] & [B_{2,1,1\epsilon_1 2}] = -[v_5 \ v_{13}] \\ [A_{2,2b3}] = [v_5 \ v_5 \ v_{10} \ v_{13} \ v_{15} \ v_{15}] & [B_{3,2b3}] = -[v_6 \ v_9 \ v_{11} \ v_{14} \ v_{17} \ v_{16}] \\ [A_{3,3\epsilon_2 2}] = [v_9 \ v_{14}] & [B_{2,3\epsilon_2 2}] = -[v_{10} \ v_{15}] \\ [A_{3,3\epsilon_3 4}] = [v_6 \ v_{11} \ v_{17} \ v_{16}] & [B_{4,3\epsilon_3 4}] = -[v_7 \ v_{12} \ v_{18} \ v_{18}] \\ [A_{4,4Y5}] = [v_7 \ v_{12} \ v_{18}] & [B_{5,4Y5}] = -[v_8 \ v_8 \ v_8] \end{pmatrix}$$

The cast S' defined by $Unite(1, 2, 6)$ is the following:



Its template is:

$$[Temp.S'] = \begin{array}{c} 0 \\ 6 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} X \\ [A_{0,0X6}] \\ [B_{6,0X6}] \\ - \\ - \\ - \end{array} \begin{array}{c} b \\ - \\ [A_{6,6b3}] \\ [B_{3,6b3}] \\ - \\ - \end{array} \begin{array}{c} \epsilon_2 \\ - \\ [B_{6,3\epsilon_26}] \\ [A_{3,3\epsilon_26}] \\ - \\ - \end{array} \begin{array}{c} \epsilon_3 \\ - \\ - \\ [A_{3,3\epsilon_34}] \\ [B_{4,3\epsilon_34}] \\ - \end{array} \begin{array}{c} Y \\ - \\ - \\ [A_{4,4Y5}] \\ [B_{5,4Y5}] \end{array} \Bigg]$$

The expression of S' over its own template is:

$$\{S'/[Temp.S']\} = \begin{pmatrix} [A_{0,0X6}] = [v_1] & [B_{6,0X6}] = -[v_7] \\ [A_{6,6b3}] = [v_7] & [B_{3,6b3}] = -[v_4] \\ [A_{3,3\epsilon_26}] = [v_4] & [B_{6,3\epsilon_26}] = -[v_7] \\ [A_{3,3\epsilon_34}] = [v_4] & [B_{4,3\epsilon_34}] = -[v_5] \\ [A_{4,4Y5}] = [v_5] & [B_{5,4Y5}] = -[v_6] \end{pmatrix}$$

10.5.1 Forward Derivation

From this situation on, the eAG modification derived from $Unite(1, 2, 6)$, that transforms I_S into a graph denoted $I_{S'}$ hereinafter and expressed over the template of S' , can be defined as follows.

In the schema field, the $Unite(1, 2, 6)$ modification implies that the two nodes of type 1 and 2, connected together by means of an ϵ_1 edge, will be merged into one new node whose type is 6, and that plays the role of summit or end for all the edges whose summit or end was of type 1 or 2.

In the instance field, $Unite(1, 2, 6)$ can be interpreted as follows: the pairs of nodes whose types are 1 et 2, connected together by an edge instantiating ϵ_1 , have to be replaced by a new node. But importantly, while there is only one pair of nodes whose types are 1 and 2 in a schema, there can be, in the instance field, several such pairs, and those pairs may even share one item. Two pairs sharing a node shall, in this case, be merged into the same node

The first thing to do, in order to derive a forward modification from $Unite(1, 2, 6)$ is to establish the list of pairs of nodes so that the first node of the pair is the summit of an instance of ϵ_1 , and the second, the end of the same instance. Second, those pairs will have to be grouped by the classes of pairs that will have to be merged into the same node. The pairs of nodes belonging to the same instance of $1[\epsilon_1]2$ are made out of the nodes whose identifiers appear on the same column of $[A_{1,1\epsilon_12}]$ and $[B_{2,1,1\epsilon_12}]$ respectively. The two pairs we get in this example are (3; 5) and (4; 13).

Since no node is shared between the two, then, they will be merged into two separate new nodes. Let us thus define two new nodes v_{20} and v_{21} of type 6, respectively associated with each of the above pairs.

The fact that both v_3 and v_5 shall be replaced by v_{20} , in the inner matrices of $[I_{S'}/[Temp.S']]$, the template-based matrix representing $I_{S'}$, and v_{14} and v_{13} by v_{21} , can be represented by a substitution matrix whose first line contains the identifier values of the nodes of I_S to be replaced and the second, the identifier value of the substitution node.

In our example, this shall give:

$$\begin{pmatrix} 3 & 4 & 5 & 13 \\ 20 & 21 & 20 & 21 \end{pmatrix}$$

Comment. Given a modification $Unite(a, b, c)$, the set of all the pairs of root and leaf of the instances of $a[\epsilon]b$ is, in general, as easy as indicated above: if $N = size_W.[A_{a,aeb}]$, then the pairs are the elements of P defined by:

$$P = \{(i, j) \mid [A_{a,aeb}]_{i,k} = 1 \wedge [B_{b,aeb}] = -1; 0 < k \leq N\}$$

Now, the fact that, in the above example, the pairs of root and leaf of the instances of $1[\epsilon_1]2$ do not intersect is a particular situation in which determining the substitution matrix is trivial.

Yet, in general, as mentioned above, two pairs of nodes can share a node – and a pair can share a node with two other pairs – etc. This will happen anytime there is a node of type a (resp. b) serving as the summit (resp. end) of more than one edge instantiating $a[\epsilon]b$.

In general, the determination of the substitution matrix demands to split P defined above into several subsets P_x , so that:

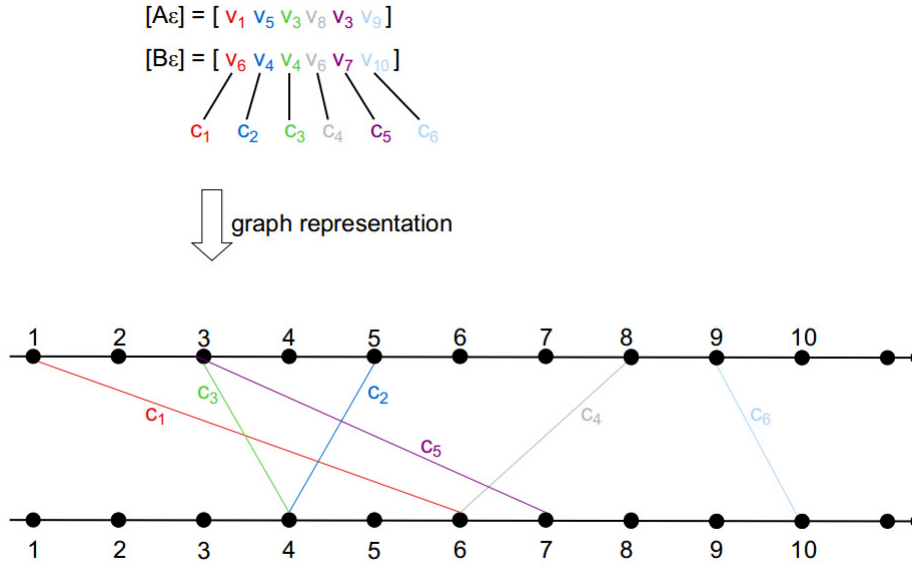
1. $\forall (i, j) \in P, \exists ! x$ so that $(i, j) \in P_x$
2. for all $P_x, (i, j) \in P$ so that $(i, j) \in P_x$,
then $\exists (i', j') \in P \mid (i = i' \vee j = j') \wedge (i, j) \neq (i', j') \Rightarrow (i', j') \in P_x$

Thus, the determination of the substitution matrix demands to find the partition of P into the maximal subsets of pairs so that the subset is either limited to one pair, or so that if a pair p_1 belongs to a subset P_x , then, there is another pair $p_2 \neq p_1$ in P_x sharing a node with p_1 .

It is worth noting that, regardless of the context, this partition of pairs of values can be obtained in linear time in terms of the number of pairs. Indeed, a pair can be represented as a non-oriented edge between two nodes of two sets (one for the first values of the pairs, one for the second ones), each associated with an identifier.

The representation of P is thus a graph, and the maximal subsets we are looking for are the maximal connected subgraphs of the (we insist: non-oriented) graph P . Let us take the following example:

Below, we represent two vectors of node identifiers $[A_\epsilon]$ and $[B_\epsilon]$ of the same size, whose n^{th} coefficients are associated into a pair c_n as highlighted below. We also provide the corresponding graph representation of $P = \{c_n\}_n$:



The graph representation of P illustrates the fact that the maximal subsets of pairs so that there is, for each pair in a set, another pair sharing one of its values, is given by the identification of the maximal connected parts of P : for instance, one such connected part is made out of the edges c_5, c_3, c_2 (indeed, in each of $((v_3; v_4), (v_3; v_7), (v_5; v_4))$ is a node that can be found in, and only in, another pair of the triple), even though c_2 and c_5 do not share any node together. Finding the set of maximal connected subgraphs in a graph can be performed in linear time in terms of the graphs's nodes (and hence, of the number of pairs in our context), e.g. by means of a graph search algorithm. Moreover, in our context (that is, when considering annotation graphs), the above graph representation of P is very easy to obtain. Since no pair can be made out of two identical values, P can be expressed by means of an incidence matrix indexed over the set of values the pairs contain for the lines, and on the c_i couples on the columns. For instance:

$$\begin{array}{c}
 1 \\
 2 \\
 3 \\
 4 \\
 5 \\
 6 \\
 7 \\
 8 \\
 9 \\
 10
 \end{array}
 \begin{bmatrix}
 c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \\
 1 & - & - & - & - & - \\
 - & - & - & - & - & - \\
 - & - & 1 & - & 1 & - \\
 - & 1 & 1 & - & - & - \\
 - & 1 & - & - & - & - \\
 1 & - & - & 1 & - & - \\
 - & - & - & - & 1 & - \\
 - & - & - & 1 & - & - \\
 - & - & - & - & - & 1 \\
 - & - & - & - & - & 1
 \end{bmatrix}$$

This matrix happens to be at hand, since it corresponds to the sum of $[A_{a,a\epsilon b}]$ and $-[B_{b,a\epsilon b}]$.

This is only natural since the general partition problem described above can, in our context, be rephrased as follows: the nodes that have to be merged together are the nodes involved in a connected subgraph of the graph expressed by $[I_S/Temp.(a\epsilon b)]$. Then, a newly created identifier value id_x is associated with each of the subsets P_x of P identified above. The ‘substitution matrix’ σ is then defined by the following algorithm by a pair of lists (σ_1 , corresponding to the first line of σ , and σ_2).

Algorithm 1: Definition of the substitution matrix.

Data: We have $\{(id_x, P_x)\}_x$ the set of newly created node identifier and pair sets defined above.

```

1 begin
2   forall the  $x$  do
3     forall the  $e \in P_x$  do
4       push. $\sigma_1(e)$  ;
5       push. $\sigma_2(id_x)$  ;

```

Once the substitution matrix is defined, $I_{S'}$ can be defined as the image of I_S by the forward derivation of $Unite(i_1, i_2, i_3)$.

Definition 10.5 : Node substitution operation. Be a substitution matrix whose coefficients belong to \mathcal{I} . Be $[M]$ whose lines are indexed on \mathcal{I} and whose columns contain only one value that is not null, and equals one, so that $[M]$ can be written $[v_a \dots v_z]$, where $\{a \dots z\} \subseteq \mathcal{I}$.

Then $\sigma.[M] = [v_{\sigma(a)} \dots v_{\sigma(z)}]$, with the following convention:

1. $\forall y \in \mathcal{I}$, if $\exists m \mid [\sigma]_{1,m} = y$, then $\sigma(y) = [\sigma]_{2,m}$
2. else $\sigma(y) = y$

Example. With the matrix values representing the graph I_S on the template of S given page 275 and the substitution matrix defined above:

- $\sigma.[B_{1,0X1}] = -\sigma.[v_3 \ v_3 \ v_4] = -[v_{20} \ v_{20} \ v_{21}]$;
- $\sigma.[A_{2,2b3}] = -\sigma.[v_5 \ v_5 \ v_{10} \ v_{13} \ v_{15} \ v_{15}] = [v_{20} \ v_{20} \ v_{10} \ v_{21} \ v_{15} \ v_{15}]$;
- $\sigma.[B_{2,3\epsilon_2}] = [v_{10} \ v_{15}] = [v_{10} \ v_{15}]$.

Hence, the above substitution mechanism enables to replace the identifier of a node involved in an instance of the ϵ edge to be merged by the identifier of the newly created node that corresponds to it. This substitution operation has to be operated on the matrices of the lines corresponding to the types whose indexes are i and j , considering $Unite(i, j, k)$, and has to be followed by the subsequent analysis in order to affect the matrix variables of $[Temp.S']$ values that describe $I_{S'}$:

We remind the reader of the template of the two schemas S and S' considered throughout this paragraph:

$$\begin{array}{l}
 [Temp.S] = \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} X \\ [A_{0,0X1}] \\ [B_{1,0X1}] \\ - \\ - \\ - \\ - \end{array} \begin{array}{c} \epsilon_1 \\ - \\ [A_{1,1\epsilon_12}] \\ [B_{2,1,1\epsilon_12}] \\ - \\ - \\ - \end{array} \begin{array}{c} b \\ - \\ - \\ [A_{2,2b3}] \\ [B_{3,2b3}] \\ - \\ - \end{array} \begin{array}{c} \epsilon_2 \\ - \\ [B_{2,3\epsilon_22}] \\ [A_{3,3\epsilon_22}] \\ - \\ - \\ - \end{array} \begin{array}{c} \epsilon_3 \\ - \\ - \\ [A_{3,3\epsilon_34}] \\ [B_{4,3\epsilon_34}] \\ - \\ - \end{array} \begin{array}{c} Y \\ - \\ - \\ - \\ [A_{4,4Y5}] \\ [B_{5,4Y5}] \end{array} \\
 \\
 [Temp.S'] = \begin{array}{c} 0 \\ 6 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} X \\ [A_{0,0X6}] \\ [B_{6,0X6}] \\ - \\ - \\ - \end{array} \begin{array}{c} b \\ - \\ [A_{6,6b3}] \\ [B_{3,6b3}] \\ - \\ - \end{array} \begin{array}{c} \epsilon_2 \\ - \\ [B_{6,3\epsilon_26}] \\ [A_{3,3\epsilon_26}] \\ - \\ - \end{array} \begin{array}{c} \epsilon_3 \\ - \\ - \\ [A_{3,3\epsilon_34}] \\ [B_{4,3\epsilon_34}] \\ - \\ - \end{array} \begin{array}{c} Y \\ - \\ - \\ - \\ [A_{4,4Y5}] \\ [B_{5,4Y5}] \end{array}
 \end{array}$$

Applying the substitution defined above to a matrix of $[I_S/[Temp.S]]$ produces another matrix value, making reference to nodes that will belong to $I_{S'}$ – but that matrix value still has to be affected to the right matrix variable of $[Temp.S']$ in order to build $I_{S'}$. The association of one matrix variable of $[Temp.S']$ to one matrix variable of $[Temp.S]$ can be done as follows:

- Given a modification $Unite(i, j, k)$, the fourth restriction in Definition 9.16 page 246 imposes that there cannot be two edges in S sharing the same label ending one on the root and the other one the leaf of the edge $i\epsilon_j$, or starting one on the root and the other one the leaf of the edge $i\epsilon_j$. This means that there is a natural correspondence between the matrix variables of $[Temp.S]$ and $[Temp.S']$: for any matrix $[M]$ from $[Temp.S]$ characterized by its sign and a triple in $\mathcal{T} \times \mathcal{L} \times \mathcal{T}$, there is one and only one matrix $[M']$ in $[Temp.S']$ that shares the same sign, the same label and either one of the two type values characteristic of $[M]$ – the other type being i or j for $[M]$ and k for $[M']$. Thus, a matrix variable $[B_{i,t \ l \ i}]$ from I_S will be associated with $[B_{k,t \ l \ k}]$ from $I_{S'}$.

For instance, in our running example, $([B_{1,0X1}]; [B_{6,0X6}])$ constitute such a couple of matrix variables.

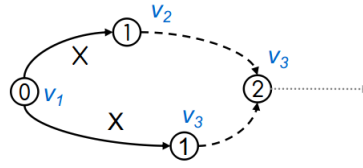
- Be $([M]; [M'])$ such a couple. Then $I_{S'}$ is defined by affecting the value $\sigma.[M]$ to $[M']$

For instance, in our example:

- $[B_{6,0X6}] = -[v_{20} \ v_{20} \ v_{21}]$;
- $[A_{6,6b3}] = [v_{20} \ v_{20} \ v_{10} \ v_{21} \ v_{15} \ v_{15}]$;
- $[B_{6,3\epsilon_26}] = [v_{10} \ v_{15}]$.

- To finish with, it has to be considered that if I_S contained patterns like the following, in which a node v_1 points towards two nodes of type i , both of them pointing towards the same node of type j , the above procedure results in redundant edges⁵.

⁵Symmetrically, a node of type i may point towards two nodes of type j , both pointing towards the same node.

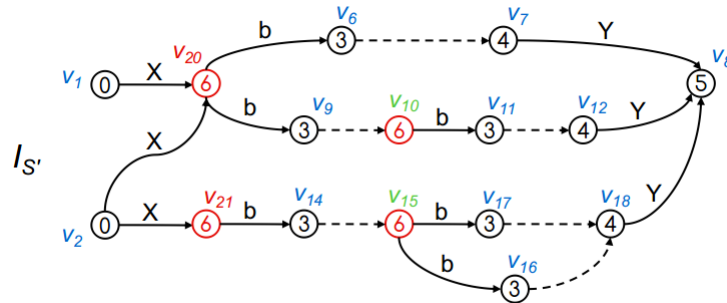


One of the redundant edges shall be deleted.

The template-based representation of the resulting graph is:

$$\{I_{S'}/[Temp.S']\} = \left(\begin{array}{ll} [A_{0,0X6}] = [v_1 \ v_2 \ v_2] & [B_{1,0X6}] = -[v_{20} \ v_{20} \ v_{21}] \\ [A_{6,6b3}] = [v_{20} \ v_{20} \ v_{10} \ v_{21} \ v_{15} \ v_{15}] & [B_{3,6b3}] = -[v_6 \ v_9 \ v_{11} \ v_{14} \ v_{17} \ v_{16}] \\ [A_{3,3\epsilon_66}] = [v_9 \ v_{14}] & [B_{6,3\epsilon_26}] = -[v_{10} \ v_{15}] \\ [A_{3,3\epsilon_34}] = [v_6 \ v_{11} \ v_{17} \ v_{16}] & [B_{4,3\epsilon_34}] = -[v_7 \ v_{12} \ v_{18} \ v_{18}] \\ [A_{4,4Y5}] = [v_7 \ v_{12} \ v_{18}] & [B_{5,4Y5}] = -[v_8 \ v_8 \ v_8] \end{array} \right)$$

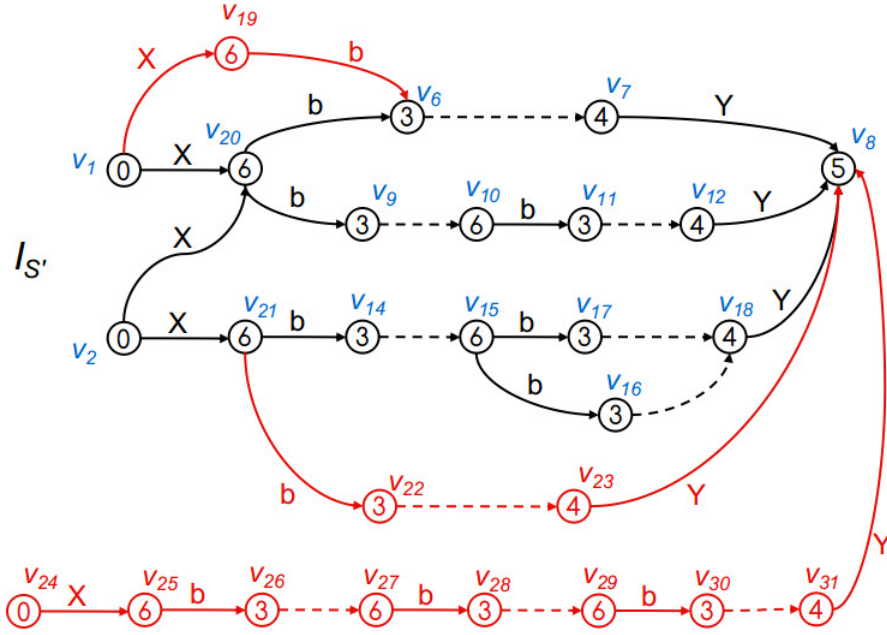
which can be illustrated as the following graph:



10.5.2 Backwards Derivation

We now present how an update on $I_{S'}$ shall be propagated to I_S . For that purpose, let us consider the update illustrated below (the red parts are added⁶):

⁶In the case of *Unite*, the kind of update that is hard to propagate is insertion, and not deletion, that simply consists in propagating the deletion on the common matrices between $[I_{S'}/[Temp.S']]$ and $[I_S/[Temp.S]]$ and/or on the matrices from both instance that are associated as described above, and from then on, identifying the nodes in the matrices describing the instances of the edge $i\epsilon j$ in $[I_S/[Temp.S]]$ that are left either as a root or a leaf in the graph. See Algorithm 3



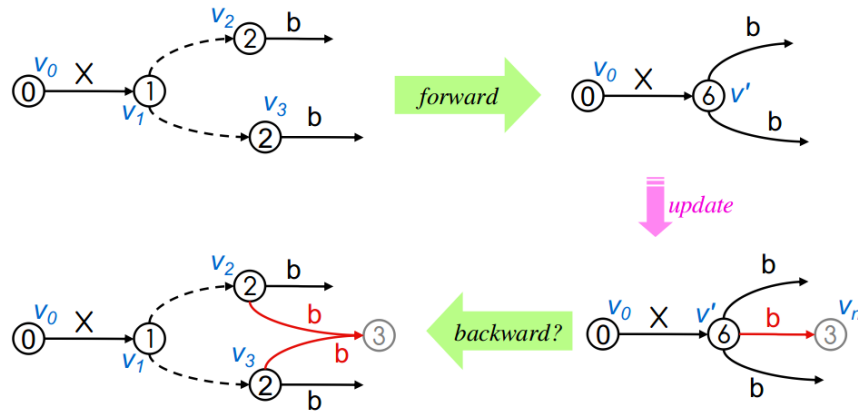
The template-based expression of this graph is the following:

$$\{I_{S'}/[Temp.S']\} = \begin{pmatrix} [A_{0,0X6}] = [v_1 \ v_2 \ v_2 \ v_1 \ v_{24}] \\ [B_{6,0X6}] = -[v_{20} \ v_{20} \ v_{21} \ v_{19} \ v_{25}] \\ [A_{6,6b3}] = [v_{20} \ v_{20} \ v_{10} \ v_{21} \ v_{15} \ v_{15} \ v_{19} \ v_{21} \ v_{25} \ v_{27} \ v_{29}] \\ [B_{3,6b3}] = -[v_6 \ v_9 \ v_{11} \ v_{14} \ v_{17} \ v_{16} \ v_6 \ v_{22} \ v_{26} \ v_{28} \ v_{30}] \\ [A_{3,3\epsilon 6}] = [v_9 \ v_{14} \ v_{26} \ v_{28}] \\ [B_{6,3\epsilon 6}] = -[v_{10} \ v_{15} \ v_{27} \ v_{29}] \\ [A_{3,3\epsilon 34}] = [v_6 \ v_{11} \ v_{17} \ v_{16} \ v_{22} \ v_{30}] \\ [B_{4,3\epsilon 34}] = -[v_7 \ v_{12} \ v_{18} \ v_{18} \ v_{23} \ v_{31}] \\ [A_{4,4Y5}] = [v_7 \ v_{12} \ v_{18} \ v_{23} \ v_{31}] \\ [B_{5,4Y5}] = -[v_8 \ v_8 \ v_8 \ v_8 \ v_8] \end{pmatrix}$$

We remind the reader that $I_{S'}$ was obtained out of I_S by means of the forward derivation of $Mod(1, 2, 6)$. The target now is to make sure to update the value of all the relevant matrices in $[I_{S'}/[Temp.S']]$ so that all the edges in $I_{S'}$ get a corresponding edge in I_S , and so that all edge b in I_S is preceded by an ϵ edge.

Special attention must be given to the nodes of $I_{S'}$ that would result from more than one pair of nodes from I_S , as illustrated below where, in I_S , $\exists e_1, e_2, v_1, v_2, v_3$ so that $v_1[e_1]v_2 \subset I_S$ and $v_1[e_2]v_3 \subset I_S$, with $type(v_1) = 1$, $type(v_2) = type(v_3) = 2$ and $label(e_1) = label(e_2) = \epsilon$.

Consider the following scenario:



In this scenario, a new node e' is created between v' the node resulting from the merging of v_1, v_2 and v_3 and another node v_n , so that this edge shall, according to the schema S , be translated into an edge starting from a node whose type is 2 in I_S . Then, what strategy to adopt for the propagation of the update to I_S ? Do we have to translate the insertion of e' into the insertion of two edges starting at v_2 and v_3 , or one edge starting at one of the two, or no edge at all?

The solution we propose is the following:

- if there is no edge $v_2[e]v_n$ or $v_3[e]v_n$ with $label(e) = label(e')$ in I_S , then the two edges are created – and it will be up to the editor to delete one, if this makes sense from an editorial point of view.
- else, the insertion is not propagated.

Algorithm. Based on the above description, we define how to propagate an update from $I_{S'}$ to I_S algorithmically. The procedure starts with the execution of Algorithm 2, that propagates the new values of all the matrices from $[I_{S'}/[Temp.S']]$ onto the corresponding matrices from $[I_S/[Temp.S]]$, apart from the ones that define the ϵ edge corresponding to the one that is merged by the forward transformation. Then, the right matrices $[A_{i_1, i_1 \epsilon i_2}]$ and $[B_{i_2, i_2 \epsilon i_2}]$ have to be written so that the propagation shall be completed. This can be done by means of the following resources:

- the non-updated couple $([A_{i_1, i_1 \epsilon i_2}] - [B_{i_2, i_2 \epsilon i_2}])$, describing the set of ϵ edges that were merged when translating I_S into $I_{S'}$. Some of those may have been deleted during the update of $I_{S'}$, which will have to be taken into account.
- the matrix σ' defined in Algorithm 2, giving the list of the new nodes of I_S having either t_{i_1} or t_{i_2} as a type value, whose insertion in I_S is due to either the insertion of new nodes having $t_{i'}$ as a type value, or to the insertion of new edges involving the pre-existing nodes typed $t_{i'}$, in $I_{S'}$.

The strategy we propose for those is to connect them to the maximum number of nodes belonging to all the nodes belonging to the same subsets of the updated

Algorithm 2: Definition of the value of the matrix variables defining I_S , after an update on I_S : first step

Data: We have :

- $[I_S/[Temp.S]]$ and $[I'_S/[Temp.S']]$, denoted $[I]$ and $[I']$ resp. hereafter ;
- $Unite(t_{i_1}, t_{i_2}, t_{i'})$, where i_1 is the index of t_{i_1} in \mathcal{T} , etc. ;
- σ , the substitution permutation defined previously.

```

1 begin
2    $\sigma' = \begin{bmatrix} \emptyset \\ \emptyset \end{bmatrix}$  ;
3   forall the  $l'$  such that  $[[I']_{i',l'}] \neq [0]$  do
4     Find the only matrix  $[[I]_{i,l}]$ ,  $i \in \{i_1; i_2\}$ , so that there is a label  $L$  and a type  $T$  such
      that  $l'$  matches the triple  $(t_{i'}, L, T)$  and  $l$  matches the triple  $(t_i, L, T)$ , or such that  $l'$ 
      matches the triple  $(T, L, t_{i'})$  and  $l$  matches the triple  $(T, L, t_i)$  ;
      /* Here we associate the submatrix corresponding to  $[[I']_{i',l'}$  in  $[I]$ . It
         could be on the block-row corresponding to  $t_{i_1}$  or  $t_{i_2}$ . The matching is
         based on the comparison of triples in  $\mathcal{T} \times \mathcal{L} \times \mathcal{T}$ , which is possible
         thanks to the restriction number 4 in the definition of the valid
         contexts for Unite. */
5     Find  $i'' \neq i', i^0 \neq i$  such that  $[[I']_{i'',l'}] \neq [0] \wedge [[I]_{i^0,l}] \neq [0]$  ;
6     Be  $[M] = [0]$ ,  $[N] = [0]$  ;
7     forall the  $(m, n)$  such that  $Abs([[I']_{i',l'}]_{m,n}) = 1$  do
8       Determine  $w_n \in \mathbb{N} \mid [[I']_{i'',l'}]_{w_n,n} \neq 0$  ;
9       Consider  $[y_1, \dots, y_N] \subseteq \sigma^{-1}(m)$  where  $\forall k, type(v_{y_k}) = t_{i'}$ , and so that
          $\forall x \in \sigma^{-1}(m) \setminus [y_1, \dots, y_N], type(v_x) \neq t_{i'}$  ;
10      if  $\exists j \in \mathbb{N}, \exists K \in [1; N] \mid \exists [[I]_{i^0,l}]_{w_n,j} \neq 0 \wedge [[I]_{i,l}]_{y_K,j} \neq 0$  then
11        forall the  $j \in \mathbb{N}, k \in [1; N], \mid [[I]_{i^0,l}]_{w_n,j} \neq 0 \wedge [[I]_{i,l}]_{y_K,j} \neq 0$  do
12           $[M]_{y_k,j} = [[I]_{i,l}]_{y_K,j}$  ;
13           $[N]_{w_n,j} = [[I']_{i'',l'}]_{w_n,n}$  ;
14        /* The previous condition in the IF instruction means that for a given
           edge in  $I_{S'}$  labelled  $L$  between a node  $v_m$  (of type  $t_{i'}$ ) and a node
            $v_{w_n}$ , we first check whether there exists an edge labelled  $L$  between
           any node of the set  $E_m$  (corresponding to  $v_m$ ) and  $v_{w_n}$  in  $I_S$ . If
           so, no new edge must be created : we simply copy the existing edges
           connected to  $v_{w_n}$ . If not, we apply the following instructions.
           */
15      else
16        /* If we get here, it means that  $v_m$  is involved, summit or end, in
           an edge of  $I_{S'}$  that has no matching in  $I_S$ . A new node  $v_{y_n}$  will
           be created to play the role of summit or end, accordingly, of a
           new edge in  $I_S$  that will fill the gap. */
17        /* Nota : at the end of the ELSE instruction, no  $\epsilon$  edge will have
           been created to connect  $v_{y_n}$  to the rest of the graph. This will
           be done in the next algorithm. */
18        Create  $v_{y_n}$  so that  $y_n \notin Id_{I_S}^V \cup Id_{I_{S'}}^V$ , and whose type matches the one of
19         $[[I]_{i,l}]$  ;
20        Determine  $j_n$  the index, in  $\mathcal{E}$ , of the edge between  $v_{y_n}$  and  $v_{w_n}$ , and whose
21        label value matches the one corresponding to  $l \in \mathcal{T} \times \mathcal{L} \times \mathcal{T}$  ;
22         $[M]_{y_n,j_n} = [[I']_{i',l'}]_{m,n}$  ;
23         $[N]_{w_n,j_n} = [[I']_{i'',l'}]_{w_n,n}$  ;
24         $push.\sigma'_1(y_n)$  ;
25         $push.\sigma'_2(m)$  ;
26       $[[I]_{i,l}] = [M]$  ;
27       $[[I]_{i^0,l}] = [N]$  ;

```

partition of the pairs of roots and edges of $t_{i_1} \epsilon t_{i_2}$, but having a different type value (that is: connecting a new summit to all the possible ends fro the same subset, or conversely).

This strategy may, for the biggest subsets of the above partition, result in a great number of edges. Our position on that point is that the editor will, if needed, clean the graph from the edges she might consider not significant.

The corresponding algorithm is Algorithm 3. To finish with, we may highlight a point that will be illustrated below. It happens that, through the forward transformation, the nodes whose type is either t_{i_1} or t_{i_2} and that are not involved in an instance of the edge $t_{i_1} \epsilon t_{i_2}$ shall be maintained in $I_{S'}$; only their type value shall be changed to $t_{i'}$. This preservation of those particular nodes means that their reference value shall be maintained through the forward modification, and afterwards synchronized as long as those nodes exist in both instance. This idea that two nodes belonging to two different instances shall be synchronized is different from the notion of ‘reference initialization’ described in Paragraph 10.6, by means of which a reference value is given to the new nodes of an instance based on the reference value of some node(s) from the other instance – without being synchronized with them.

In Algorithm 1, this synchronization is not done. Indeed, in the first part of the algorithm, the nodes v' from $I_{S'}$ whose type value is $t_{i'}$ are split into two cases: they either result from the merging of an ϵ edge through the forward transformation, in which case the condition on Line 10 is verified; otherwise, they are involved in edges that were created in $I_{S'}$ during an update, and some new node(s) may be created in I_S .

Yet, because the degree of the nodes of type t_{i_1} or t_{i_2} can be more than 1 for the operation of *Unite*, some nodes whose type is $t_{i'}$ might not result from the merging of any ϵ edge, but instead be involved in edges that are shared, or that correspond, in the two instances – in which case v' and its corresponding node in I_S , whose type value will be either t_{i_1} or t_{i_2} , shall always have the same reference value.

To distinguish between the nodes v' that do not result from the merging of an instance of $t_{i_1} [\epsilon] t_{i_2}$, we can use the following characteristics: if m is the identifier of v' , then all the v_j nodes from I_S so that $\sigma(j) = m$ have the same type value (either t_{i_1} or t_{i_2}). In this case, we shall simply replace, during the backwards propagation, v_j by v' in the corresponding matrices, as indicated in Algorithm 4.

Illustration. Let us run the above algorithms on our example.

With the above nomenclature, in this example, $t_{i_1} = 1$, $t_{i_2}, t_{i'} = 6$; $[I] = [I_S/[Temp.S]]$, $[I'] = [I_{S'}/[Temp.S']]$.

First part of the algorithm. σ' is initialized as a pair of empty lists.

Let us consider the values l' so that $[[I']_{i',l'}] \neq [0]$.

First, $l' = 1$, which makes reference to $[[I']_{2,1}] = [B_{6,0X6}] = -[v_{20} \ v_{20} \ v_{21} \ v_{19} \ v_{25}]$.

1. The block-column of index l' from $[I']$ is characterized by the triple $(0; X; 6)$ in $\mathcal{T} \times \mathcal{L} \times \mathcal{T}$. The only inner matrix from $[I]$ characterized by $(0; X; i)$ with $i \in \{i_1; i_2\}$ is $[[I]_{2;1}] = [B_{1,0X1}] = -[v_3 \ v_3 \ v_4]$.

Algorithm 3: Definition of the value of the matrix variables defining I_S , after an update on $I_{S'}$: second step.

Data: We have :

- $[I_S/[Temp.S]]$ and $[I'_S/[Temp.S']]$, denoted $[I]$ and $[I']$ resp. hereafter ;
- $[A_{i_1, i_1 \epsilon i_2}]$ and $[B_{i_2, i_1 \epsilon i_2}]$, denoted $[A_\epsilon]$ and $[B_\epsilon]$ hereafter ;
- $Unite(t_{i_1}, t_{i_2}, t_{i'})$, where i_1 is the index of t_{i_1} in \mathcal{T} , etc. ;
- σ , the substitution permutation defined previously;
- σ' , defined in the previous algorithm.

```

1 begin
2   Create  $[A] = [0]$  ;
3   Create  $[B] = [0]$  ;
4   /* ----- First part : ----- */
5   forall the  $a, b, n \in \mathbb{N} \mid [A_\epsilon]_{a,n} \neq [0] \wedge [B_\epsilon]_{b,n} \neq [0]$  do
6     if  $\exists j_a, j_b, n_a, n_b \mid [[I]_{i_1, j_a}]_{a, n_a} \neq 0 \wedge [[I]_{i_2, j_b}]_{a, n_b} \neq 0$ , with  $[[I]_{i_1, j_a}] \neq [A_\epsilon]$  and
7      $[[I]_{i_2, j_b}] \neq [B_\epsilon]$  then
8        $[A]_{a,n} = [A_\epsilon]_{a,n}$  ;
9        $[B]_{b,n} = [B_\epsilon]_{b,n}$  ;
10      /* This means all the  $\epsilon$  edges from the former version of  $I_S$  are
11      maintained, if both their summit and end appear in at least one
12      matrix of their respective line-block. */
13    else
14      forall the  $k \mid [\sigma_1]_k \in \{v_a, v_b\}$  do
15        Delete  $[\sigma_1]_k$  and  $[\sigma_2]_k$  ;
16        /* Since the substitution node has been deleted in  $I_{S'}$ , the
17        original nodes do not appear anymore neither in  $I_S$ , nor in  $\sigma$  -
18        hence the operation above. */
19      /* ----- Second part : ----- */
20      forall the  $m$  such that  $\exists y \mid \sigma'(y) = m$  do
21        Build  $E'_m = \{v_y \mid \sigma'(y) = m\}$  ;
22        Build  $E_m^0 = \{v_y \mid \sigma(y) = m\}$  ;
23        /* In the line above,  $\sigma$  is considered in its current state, that is, as
24        updated at the end of the first half of the current algorithm. */
25        Be  $E_m = E_m^0 \cup E'_m$  ;
26        forall the  $v_y \in E'_m$  do
27          Determine  $type(v_y)$  ;
28          forall the  $v_i \in E_m \mid type(v_i) \neq type(v_y)$  do
29            if  $type(v_y) = t_{i_1}$  then
30              Determine  $j$  the index in  $\mathcal{E}$  of the edge  $e$  labelled  $\epsilon$  and so that
31               $sut(e) = v_y$  and  $end(e) = v_i$  ;
32               $[A]_{y,j} = 1$  ;
33               $[B]_{i,j} = -1$  ;
34            else
35              Determine  $j$  the index in  $\mathcal{E}$  of the edge  $e$  labelled  $\epsilon$  and so that
36               $sut(e) = v_i$  and  $end(e) = v_y$  ;
37               $[A]_{i,j} = 1$  ;
38               $[B]_{y,j} = -1$  ;
39          end
40        end
41      end
42       $[A_\epsilon] = [A]$  ;
43       $[B_\epsilon] = [B]$  ;
44      Redefine  $\sigma$  as the concatenation of the previous version of  $\sigma$  and  $\sigma'$  ;

```

Algorithm 4: Definition of the value of the matrix variables defining I_S , after an update on $I_{S'}$: last step

Data: We have :

- $[I_S/[Temp.S]]$ and $[I_{S'}/[Temp.S']]$, denoted $[I]$ and $[I']$ resp. hereafter ;
- σ , the updated substitution permutation ;
- σ' , the partial substitution permutation built in the first part of the algorithm.

```

1 begin
2   forall the  $m \in [\sigma']_2 \setminus [\sigma]_2$  do
3     if  $\forall v_K, v_L \in E'_m, type(v_K) = type(v_L)$  then
4       Be  $i$  the index in  $\mathcal{T}$  of any  $v_M \in E'_m$  ;
5       forall the  $\forall k \in \sigma'^{-1}(m), n, j \mid [[I]_{i,n}]_{k,j} \neq 0$  do
6          $[[I]_{i,n}]_{m,j} = [[I]_{i,n}]_{k,j}$  ;
7          $[[I]_{i,n}]_{k,j} = 0$  ;

```

2. The matrices that complement the description of the edges labelled X coming from nodes typed 0, in $[I']$ and $[I]$ are the inner matrices with $(i'' = 1; l' = 1)$ and $(i^0 = 1; l = 1)$, that is $[[I']_{1,1}] = [A_{0,0X6}] = [v_1 \ v_2 \ v_2 \ v_1 \ v_{24}]$ and $[[I]_{1,1}] = [A_{0,0X1}] = [v_1 \ v_2 \ v_2]$ respectively.
3. We define $[M] = [0]$, $[N] = [0]$.
4. We will now consider the non-null coefficients from $[[I']_{i',l'}] = [[I']_{2,1}] = -[v_{20} \ v_{20} \ v_{21} \ v_{19} \ v_{25}]$.
 - (a) Let us consider $[[I']_{i',l'}]_{20,1}$. The index w_n so that $[[I']_{i'',l'}]_{w_n,1} \neq 0$ is $w_n = 1$.
 - (b) Now we note that $\sigma^{-1}(20) = \{3; 5\}$. Given we are interested in propagating the nodes from $[[I']_{1,1}]$ into an inner matrix of I_S that corresponds to the type value 1, and since the only node, between v_3 and v_5 , whose type value is v_3 ($type(v_5) = 2$), then the set denoted $[y_1 \dots y_N]$ in the algorithm contains one single value, 3, at this step of the algorithm.
 - (c) We then search whether there is a value j so that $[[I]_{1,1}]_{1,j} \neq 0 \wedge [[I]_{2,1}]_{3,j} \neq 0$. It happens that $[[I]_{1,1}]_{1,1} = 1$ and $[[I]_{2,1}]_{3,1} = -1$. Thus $j = 1$.
 - (d) Hence the following affectations:
 - $[M]_{3,1} = [[I]_{2,1}]_{3,1} = -1$, that is $[M] = -[v_3]$;
 - $[N]_{1,1} = [[I]_{1,1}]_{1,1} = 1$, that is $[N] = [v_1]$.
 - (e) The above procedure runs exactly the same way for $[[I']_{i',l'}]_{20,2}$ and $[[I']_{i',l'}]_{21,3}$, which results in $[M] = -[v_3 \ v_3 \ v_5]$ and $[N] = [v_1 \ v_2 \ v_2]$.
 - (f) On the contrary, when analysing $[[I']_{i',l'}]_{19,4}$, it happens that $\sigma^{-1}(19) = []$. Then the IF on Line 10 of Algorithm 2 is not verified. Hence the following procedure:
 - We identify $w_n = 1$ so that $[[I']_{i'',l'}]_{w_n,4} = 1$.
 - A new identifier y_n , belonging to none of the sets of existing identifiers of I_S and $I_{S'}$, is created, for instance $y_n = 40$.
 - The index j_n in \mathcal{I} of the edge connecting v_{40} of type 1 to v_1 of type 0, and labelled X , should be determined; yet the value of this index

does not really matter. It is fine to simply insert the new values below at the end of the matrices, in an additional, last column, and the value $j_n = size_W([M]) + 1 = size_W([N]) + 1 = 4$ is acceptable;

- Then the matrices are given the following additional values: $[M]_{40,4} = -1$ and $[N]_{1,4} = 1$;
- σ' is given the following value:

$$\sigma' = \begin{bmatrix} [40] \\ [19] \end{bmatrix}$$

(g) The above procedure runs exactly the same for $[[I']_{i',l'}]_{25,5}$:

- We identify $w_n = 24$ so that $[[I']_{i'',l'}]_{w_n,4} = 1$.
- A new identifier y_n is created: for instance, $y_n = 42$.
- Idem, $j_n = size_W([M]) + 1 = size_W([N]) + 1 = 5$;
- The matrices $[M]$ and $[N]$ are given the following values: $[M]_{42,5} = -1$ and $[N]_{24,5} = 1$;
- σ' is updated:

$$\sigma' = \begin{bmatrix} [40 & 42] \\ [19 & 25] \end{bmatrix}$$

The resulting matrix values are the following:

- $[[I]_{2,1}] = [M] = -[v_3 \ v_3 \ v_5 \ v_{40} \ v_{42}]$;
- $[[I]_{1,1}] = [N] = [v_1 \ v_2 \ v_2 \ v_1 \ v_{24}]$.

The above procedure is repeated for $l' = 2$, that is for:

$$[[I']_{i',l'}] = [[I']_{2,2}] = [A_{6,6b3}] = [v_{20} \ v_{20} \ v_{10} \ v_{21} \ v_{15} \ v_{15} \ v_{19} \ v_{21} \ v_{25} \ v_{27} \ v_{29}]$$

1. The block-column of index l' , in $[I']$, is characterized by the triple $(6; b; 3)$ in $\mathcal{T} \times \mathcal{L} \times \mathcal{T}$. The only inner matrix from $[I]$ characterized by a triple $(i; b; 3)$ with $i \in \{i_1; i_2\}$ is $[[I]_{3,3}] = [A_{2,2b3}] = [v_5 \ v_5 \ v_{10} \ v_{13} \ v_{15} \ v_{15}]$.
2. The matrices that complement the description of the instances of the edge $6b3$, in $[I']$ and $[I]$, are the inner matrices with the indexes $(i'' = 3; l' = 2)$ and $(i^0 = 4; l = 3)$, that is $[[I']_{3,2}] = [B_{3,6b3}] = -[v_6 \ v_9 \ v_{11} \ v_{14} \ v_{17} \ v_{16} \ v_6 \ v_{22} \ v_{26} \ v_{28} \ v_{30}]$ et $[[I]_{4,3}] = [B_{3,2b3}] = -[v_6 \ v_9 \ v_{11} \ v_{14} \ v_{17} \ v_{16}]$ respectively.
3. We initialize $[M] = [0]$, $[N] = [0]$.
4. Let us now consider the non-null coefficients from $[[I']_{i',l'}] = [[I']_{2,2}] = -[v_{20} \ v_{20} \ v_{10} \ v_{21} \ v_{15} \ v_{15} \ v_{19} \ v_{21} \ v_{25} \ v_{27} \ v_{29}]$.

(a) For the six first values from $[[I']_{i',l'}]$, one can check that the algorithm runs similarly as above for $[[I']_{2,1}]_{20,1}$

Hence the following intermediary matrix values:

- $[M] = [v_5 \ v_5 \ v_{10} \ v_{13} \ v_{15} \ v_{15}]$;
- $[N] = -[v_6 \ v_9 \ v_{11} \ v_{14} \ v_{17} \ v_{16}]$.

σ' is not modified at this step.

(b) For the seventh value from $[[I']_{i',l'}]$, the situation is the same as for $[[I']_{2,1}]_{19,4}$: $\sigma^{-1}(19) = []$. The same procedure as above provides the following values:

- $[M] = [v_5 \ v_5 \ v_{10} \ v_{13} \ v_{15} \ v_{15} \ v_6]$;
- $[N] = -[v_6 \ v_9 \ v_{11} \ v_{14} \ v_{17} \ v_{16} \ v_{41}]$.

σ' is updated:

$$\sigma' = \begin{bmatrix} [40 \ 42 \ 41] \\ [19 \ 25 \ 19] \end{bmatrix}$$

(c) Let us then consider $[[I']_{i',l'}]_{21,8}$.

Here, the value of w_n so that $[[I']_{i'',l'}]_{w_n,8} \neq 0$ is $w_n = 22$. The situation now is different from the situations described so far: the node v_m corresponding to $[[I']_{i',l'}]_{21,8}$ results from the merging of a pair of nodes from I_S , while v_{w_n} , that is, the node that is the end of an edge originating in v_m in $I_{S'}$, does not.

The procedure is the following:

- A new identifier is created, for instance $y_n = 39$.
- $j_n = size_W([M]) + 1 = size_W([N]) + 1 = 8$;
- $[M]_{39,8} = 1$ and $[N]_{22,8} = -1$;
- σ' is updated:

$$\sigma' = \begin{bmatrix} [40 \ 42 \ 41 \ 39] \\ [19 \ 25 \ 19 \ 21] \end{bmatrix}$$

(d) Without further detail, for $(m, n) = (25, 9)$, we get:

- $[M]_{43,9} = 1$ and $[N]_{26,9} = -1$;
- $\sigma' = \begin{bmatrix} [40 \ 42 \ 41 \ 39 \ 43] \\ [19 \ 25 \ 19 \ 21 \ 25] \end{bmatrix}$

(e) To finish with, one more comment. We now consider the tenth value of $[[I']_{i',l'}]$, that designates the node v_{27} . In the following, it will appear that the node corresponding to this node in I_S will not take part to an instance of $1|\epsilon|2$, which will be handled by Algorithm 4.

- $w_n = 28$.
- A new identifier is created, for instance, $y_n = 50$.
- $j_n = size_W([M]) + 1 = size_W([N]) + 1 = 10$;
- Then $[M]_{50,10} = -1$ and $[N]_{28,10} = -1$;
- σ' is updated:

$$\sigma' = \begin{bmatrix} [40 \ 42 \ 41 \ 39 \ 43 \ 50] \\ [19 \ 25 \ 19 \ 21 \ 25 \ 27] \end{bmatrix}$$

(f) Idem, $[[I']_{i',l'}]_{29,11}$ gives the following values :

- $w_n = 30$.
- A new identifier is created, for instance $y_n = 51$.
- $j_n = size_W([M]) + 1 = size_W([N]) + 1 = 10$;
- $[M] = [v_5 \ v_5 \ v_{10} \ v_{13} \ v_{15} \ v_{15} \ v_{41} \ v_{39} \ v_{43} \ v_{50} \ v_{51}]$
- $[N] = -[v_6 \ v_9 \ v_{11} \ v_{14} \ v_{17} \ v_{16} \ v_6 \ v_{22} \ v_{26} \ v_{28} \ v_{30}]$;
- σ' is updated:

$$\sigma' = \begin{bmatrix} [40 \ 42 \ 41 \ 39 \ 43 \ 50 \ 51] \\ [19 \ 25 \ 19 \ 21 \ 25 \ 27 \ 29] \end{bmatrix}$$

Second part of the algorithm. Let us denote $[A_\epsilon] = [A_{1,1\epsilon 2}] = [v_3 \ v_4]$ and $[B_\epsilon] = [B_{2,1\epsilon 2}] = -[v_5 \ v_{13}]$.

Be two null matrices $[A]$ and $[B]$.

1. Be $n = 1$. The condition $[A_\epsilon]_{a,n} = 1 \wedge [B_\epsilon]_{b,n} = 1$ is verified for $a = 3$ and $b = 5$.
 - (a) We then look for an inner matrix of the block-line i_1 of $[I]$ where v_a shall appears. This search give the following result: the first column of $[B_{1,0X1}]$ describes v_3 .
 - (b) We then look for an inner matrix of the block-line i_2 of $[I]$ where v_b shall appears. This search give the following result: the first column of $[A_{2,2b3}]$ describes v_5 .
 - (c) Thus, at this step: $[A] = [v_3]$ et $[B] = -[v_5]$.
2. Idem for $n = 2$:
 - (a) $[A] = [v_3 \ v_4]$
 - (b) $[B] = -[v_5 \ v_{13}]$.
3. We then consider σ' , whose value is:

$$\sigma' = \begin{bmatrix} [40 \ 42 \ 41 \ 39 \ 43 \ 50 \ 51] \\ [19 \ 25 \ 19 \ 21 \ 25 \ 27 \ 29] \end{bmatrix}$$

4. To start with, we build $E'_{19} = \{v_j \mid \sigma'(j) = 19\} = \{v_{40}; v_{41}\}$ and $E_{19}^0 = \{v_j \mid \sigma(j) = 19\} = \{\}$.
5. Here, $E_m = E'_m$.
6. The values from E'_m are then considered sequentially. Let us consider v_{40} .
 - v_{40} appears in $[B_{1,0X1}]$. Its type is thus 1.
 - The only element from E_m whose type value differs from $type(v_{40})$ is v_{41} .
 - Thus an ϵ edge must be instantiated between v_{40} and v_{41} :

$$\begin{aligned} [A]_{40,1} &= 1 ; \\ [B]_{41,1} &= -1. \end{aligned}$$

7. Let us then consider $m = 25$. The situation is identical as for $m = 19$:

$$\begin{aligned} [A]_{42,1} &= 1 ; \\ [B]_{43,1} &= -1. \end{aligned}$$

8. The situation is slightly different for $m = 21$. Indeed, $E'_{21} = \{39\}$ and $E_{21}^0 = \{4; 13\}$. An edge ϵ must be created between v_{39} and all the nodes from E_{21}^0 whose type is different from $type(v_{39})$. Only v_4 is concerned here. At the end of this step, we have:

$$\begin{aligned} [A] &= [v_3 \ v_4 \ v_{40} \ v_{42} \ v_4] ; \\ [B] &= -[v_5 \ v_{13} \ v_{41} \ v_{43} \ v_{39}]. \end{aligned}$$

9. Let us then consider $m = 27$. $E'_m = \{v_{50}\} = E_m$. There is no node in E_m whose type is different from the type of v_{50} , so the procedure ends here for this value of m . This is consistent with the fact that v_{50} is not meant to be part of an instance of the edge $1|\epsilon|2$.

10. The same applies for $m = 29$.

Hence:

$$\begin{aligned} [A_\epsilon] &= [v_3 \ v_4 \ v_{40} \ v_{42} \ v_4] ; \\ [B_\epsilon] &= -[v_5 \ v_{13} \ v_{41} \ v_{43} \ v_{39}]. \end{aligned}$$

Third part of the algorithm. To finish with, we have to deal with the nodes like v_{50} , that do not belong to an instance of $1[\epsilon]2$.

1. First, we must identify the values m appearing in the second line of σ' and missing in the second line of σ : those are $m \in \{19; 25; 27; 29\}$.
2. Let us consider $m = 19$ first.

We have seen that $E'_{19} = \{40; 41\}$. The condition $\forall v_j, v_k \in E'_m, type(v_j) = type(v_k)$ n'is not verified. The procedure stops here for this value of m .

3. The same applies for $m = 25$.
4. Let us then consider $m = 27$.

We have seen that $E'_{27} = \{v_{50}\}$. In this case, the condition $\forall v_j, v_k \in E'_m, type(v_j) = type(v_k)$ is verified.

- $type(v_{50}) = 2$, and this type value has the index 3 in \mathcal{T} .
- Then for all $v_j \in E'_m$, that is here for v_{50} only, we have to find the set of values n, j so that $[[I]_{3,n}]_{50,j} \neq 0$. Here: $n = 3$ et $j = 10$, since $[A_{2,2b3}]_{50,10} = 1$ is the only solution for the inequality above.
- Hence:

$$\begin{aligned} [A_{2,2b3}]_{m,10} &= [A_{2,2b3}]_{27,10} = 1 ; \\ [A_{2,2b3}]_{50,10} &= 0. \end{aligned}$$

5. The same applies for $m = 29$, which gives:

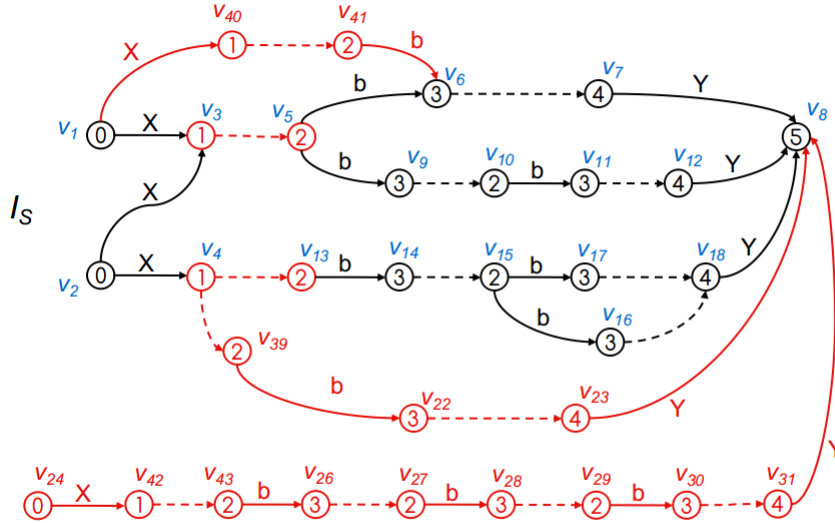
$$\begin{aligned} [A_{2,2b3}]_{m,11} &= [A_{2,2b3}]_{29,11} = 1 ; \\ [A_{2,2b3}]_{51,11} &= 0. \end{aligned}$$

In fine, the above algorithm gives:

$$\{I_{S'}/[Temp.S']\} =$$

$$\left(\begin{array}{l} [A_{0,0X1}] = [v_1 \ v_2 \ v_2 \ v_1 \ v_{24}] \\ [B_{1,0X1}] = -[v_3 \ v_3 \ v_4 \ v_{40} \ v_{42}] \\ [A_{1,1\epsilon_1 2}] = [v_3 \ v_4 \ v_4 \ v_{40} \ v_{42}] \\ [B_{2,1\epsilon_1 2}] = -[v_5 \ v_{13} \ v_{39} \ v_{41} \ v_{43}] \\ [A_{2,2b3}] = [v_5 \ v_5 \ v_{10} \ v_{13} \ v_{15} \ v_{15} \ v_{41} \ v_{39} \ v_{43} \ v_{27} \ v_{29}] \\ [B_{3,2b3}] = -[v_6 \ v_9 \ v_{11} \ v_{14} \ v_{17} \ v_{16} \ v_6 \ v_{22} \ v_{26} \ v_{28} \ v_{30}] \\ [A_{3,3\epsilon_2 2}] = [v_9 \ v_{14} \ v_{26} \ v_{28}] \\ [B_{2,3\epsilon_2 2}] = -[v_{10} \ v_{15} \ v_{27} \ v_{29}] \\ [A_{3,3\epsilon_3 4}] = [v_6 \ v_{11} \ v_{17} \ v_{16} \ v_{22} \ v_{30}] \\ [B_{4,3\epsilon_3 4}] = -[v_7 \ v_{12} \ v_{18} \ v_{18} \ v_{23} \ v_{31}] \\ [A_{4,4Y5}] = [v_7 \ v_{12} \ v_{18} \ v_{23} \ v_{31}] \\ [B_{5,4Y5}] = -[v_8 \ v_8 \ v_8 \ v_8 \ v_8] \end{array} \right)$$

which can be represented as follows:



10.6 Reference Values Propagation

In the above paragraphs, we introduced how a transformation between two schemas can be interpreted into a bidirectional transformation in the instance field, that is, a transformation translating a graph expressible on the template of the first schema into a graph expressible on the template of the second schema. The propositions formulated above focus on the structural aspects of this transformation. The question of the reference value of the nodes of the graphs, that are meant to represent an annotation, has been left aside. This is the point we discuss here.

Each node of a graph, in the instance field, has to be affected a reference value associated with a predetermined chronology: this value “positions” the nodes delimiting each edge of the annotation graph onto the annotated resources, so that (roughly speaking) the content shall be qualified as indicated by the label of the edge. It is thus crucial that the nodes of a newly created instance graph shall be given a reference value that makes sense, from an editorial point of view – or if this is not possible automatically, that the editor shall be invited to do so.

Be then two casts S, S' and t a modification so that $S' = t.S$

Be $(I_S, I_{S'})$ le couple of graphs instantiating (S, S') , and synchronized together by the eAG modification derived from t .

The reference values are managed as follows:

1. Any nodes having the same identifier in I_S and $I_{S'}$ will have the same reference value;
2. If t is defined as $Split(i, j, k)$:
 - Be σ the substitution matrix associating the identifier of the nodes of type i and the corresponding nodes of types j, k ;
 - Be then three identifiers i_0, j_0, k_0 so that $\sigma(j_0) = \sigma(k_0) = i_0$;
 - If the three nodes are so that there is a propagated update in the forward direction (from I_S to $I_{S'}$) that implies the creation of v_{j_0} and v_{k_0} out of v_{i_0} , then the reference value of the two newly created nodes will be initialized with $ref(v_{i_0})$;
 - If the three nodes are so that $ref(v_{j_0}) = ref(v_{k_0})$ and so that there is a propagated update in the backward direction (from $I_{S'}$ to I_S) that implies the creation of v_{i_0} out of v_{j_0} and v_{k_0} , then the reference value of the newly created node will be initialized with $ref(v_{k_0})$.
3. If t is defined as $Unite(i, j, k)$:
 - Be σ the substitution matrix associating the identifier of the nodes of type i, j and the corresponding nodes of type k ;
 - Be then three identifiers i_0, j_0, k_0 so that $\sigma(i_0) = \sigma(j_0) = k_0$;
 - If the three nodes are so that $ref(v_{i_0}) = ref(v_{j_0})$ and so that there is a propagated update in the forward direction (from I_S to $I_{S'}$) that implies the creation of v_{k_0} out of v_{i_0} and v_{j_0} , then the reference value of the newly created node will be initialized with $ref(v_{i_0})$
 - If the three nodes are so that there is a propagated update in the backward direction (from $I_{S'}$ to I_S) that implies the creation of v_{i_0} and v_{j_0} out of v_{k_0} , then the reference value of the two newly created nodes will be initialized with $ref(v_{k_0})$.
4. If t is defined by $Mod(\chi^-, \chi^+)$:
 - Be (r, l) a pair of nodes playing the role of root and leaf for an instance of χ^- in I_S .
 - If there is a subgraph of $I_{S'}$, rooted in r and possessing l as its only leaf, instantiating a certain path of χ^+ whose label sequence, ϵ edges set apart, is the same as the one of a subgraph of I_S , rooted in r and possessing l as its only leaf and instantiating a given path of χ^- , then the reference value of the summits and ends of the corresponding edges of each sequence will be initialized with the same value.
 - The reference value of the other nodes will be left to the editor.
 - Symmetrically for the backwards transformation from $I_{S'}$ to I_S .

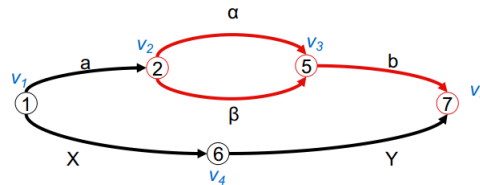
Importantly, apart from the point 1 that has to be verified anytime, the above considerations are initialization rules and only apply to the reference value of newly created nodes, that is, either when $I_{S'}$ is initialized or on the new nodes resulting from the propagation of an update. If the value of a node that does not have the same identifier in both instances is changed in one of the instances: 1) this change is not considered to be an update in the sense given in Paragraph 10.2 and 2) if this change is followed by an update, the propagation of that update will not affect the reference value of the corresponding node(s) in the other instance.

10.7 Composing Modifications: Two Quick Examples

As a conclusive remark, after the definition of the elementary modifications, let us give a quick illustration about how some transformations can be defined by composing several modifications

10.7.1 First Example.

It is highly possible that an editor might modify, in the sense of the *Mod* operator, a subgraph of a cast that does not match the definition of a cell. For instance, the following:

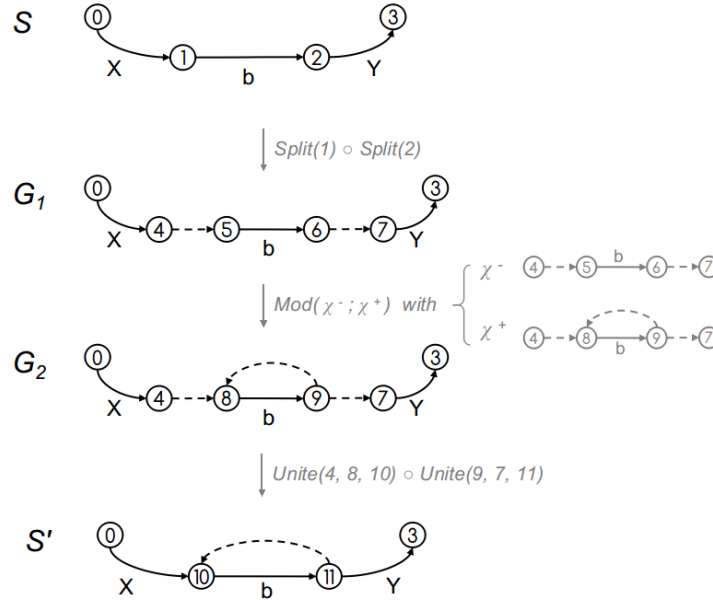


Let us suppose that the editor wants to replace the red subgraph above by a single edge X . The procedure to do such an amendment is the following

- A first elementary *Mod* operation is performed, with the linear subgraph of S located between the nodes 2 and 7, containing the edges α and b , as χ^- , and a substitution cell limited to an edge $2X'7$. The schematic cell χ_1^- is partially independent: the edge labelled X' will be inserted between the nodes 2 and 7; the edge labelled α will be deleted and the edge b maintained.
- A second modification is performed, with the linear subgraph containing the edges β et b as χ^- and a substitution cell limited to an edge $2X''7$. Since χ^- is independent, then it is replaced as a whole by the edge X'' .
- A last modification *Mod* is performed with the graph made out of the edges X and X' as χ^- and a single edge X as χ^+ .

10.7.2 Second Example: Making Some Pattern Cyclic

Cyclization is the process by which a linear subgraph originally describing a regular expression XbY (where b, X and Y are regular expressions) into another graph describing Xb^*Y . The more the cyclization procedure makes use of the *Mod* operator, that translates sequences of edges into sequences of edges, potentially with significant reference value propagation, the more information from the original instance is kept in the new instance. We propose the following procedure:



10.8 Conclusion

In this chapter, we have introduced several operators for modifying a SeAG schema S and turning it into another one. This is the meaning we give to the notion of SeAG amendment introduced in the definition of the general problematic of this dissertation. We have also shown how those operators could be interpreted in the field of the instances, and how a bidirectional transformation could be derived from a SeAG transformation, in order to provide, semi-automatically, an instance of S into a graph expressed in the template of the amended schema, and to synchronize this graph with the first instance.

Several points of our proposition shall need improvements. First, some properties have to be formally investigated: in particular, it has to be considered whether a SeAG transformation that changes a well-formed schema into another well-formed schema can be derived into an eAG transformation maintaining well-formedness as well, or not. The well-behavedness of the eAG bidirectional transformations also has to be

established. Second, the derivation of an eAG *Mod* modification from a SeAG operation has to be extended to the general case where schematic cells are unrestricted. Eventually, the complexity of the algorithms given above has to be evaluated: in a real-world application, the time-performance of those will be important, since they play a synchronizing role in the editorial system we propose.

Still, at this point, we have defined what appear to be the basis for the first application of the Data Exchange problem to multistructured data – and proposed the stems for the first (to the best of our knowledge) symmetric bidirectional transformations for (cyclic) graph-structured data.

As a conclusive remark, we would like to discuss briefly the semi-automatic quality of the instance transformations. The compound term “semi-automatic” refers to the fact that an editor may have to intervene (either by hand, or by means of any assistance tool that might have been developed independently from the system we propose) on a graph produced by the forward transformation derived from a SeAG transformation, in three different ways. First, when the reference value propagation defined in Paragraph 10.6 leaves some new nodes without a value, positioning those nodes onto the annotated resources is left to the editor. Second, when the schema transformation consists in permitting a pattern to be repeated (as described briefly in Paragraph 10.7.2), the graph corresponding to the new schema reflects the annotation that is contained in the instance of the original schema: it does not, initially, contain several occurrences in a row of the repeatable pattern. It will be up to the editor to identify where repeated occurrences are needed. Eventually, the propagation of an insertion in one instance may result in the insertion of many edges, among which some may be useless from an editorial point of view, in the other instance: for instance, several ϵ edges connecting the nodes from the same partition set P_x may result from the backward derivation of a *Unite* operation. Here again, cherry picking among all those candidate edges is left to the editor.

As indicated in the general introduction of this dissertation, the purpose of bidirectionalizing eAG/SeAG is to support collaborative data structure definition (and re-definition). The scenario we proposed to explain how to collaboratively drive the evolution of a data structure was the following: from a situation in which there is a consensual data structure S , that is instantiated by all the editors (I_S represents the collective edition), one editor A , based on editorial considerations, defines an alternative data structure S' . The eAG/SeAG bidirectionalization ensures that, as soon as S' has been defined, a graph instantiating it, and retailing as much information from I_S as possible, is created, and synchronized with I_S , so that any update on one side is propagated on the other side. Yet, as indicated above, this translation from one side to the other is semi-automatic. which means that the editor A may have to do some work (positioning some new elements onto the data, etc.) before having a meaningful instance of S' at hand.

It is also worth noting that, as soon as the editor A has defined S' , in the scenario we propose, she may start communicating with the other editors about the amendment she proposes, and illustrate the relevance of this amendment based on selected examples. It is possible that, convinced on the basis of those examples, the rest of the team shall decide that the amendment is reasonable, and shall consider choosing S' as the updated collective data structure.

Two comments can be made at this point:

1. It is possible to provide the editor A , as soon as she defines S' (actually, after $I_{S'}$ has been calculated) with a minimal quantitative evaluation of the editorial interventions needed for $I_{S'}$ to be meaningful: the number of newly created nodes without a significant reference value. Even though this number does not translate easily into an evaluation of the minimal working-time needed for making $I_{S'}$ editorially meaningful, it still provides an indication of the number of elements whose content will have to be defined.
This indication shall be given not only to A , but to all the editors discussing the relevance of the amendment proposed by A : indeed, some modifications may be interesting and, at the same time, may require too much work to be performed. In other words, opting for a new data structure not only requires to be able to evaluate the scientific relevance of the proposed amendment, but also its feasibility. Thus, if the quantitative evaluation of the minimal work is too high, an amendment may be discarded, or postponed if the agenda is not right for it to be performed at a given moment.
2. On the positive side, the quantitative evaluation of the work to be done enables the editor A not to have to do all the editorial work, at the scale of the whole corpus, to convince the others of the feasibility of the amendment: some examples, added to the quantitative evaluation, if that last is low enough, provide a good basis for the decision to change the data structure or not.

Part V

General conclusion

Conclusion

10.9 Contributions

This PhD work started by the consideration of the nature of Digital Scholarly Edition (DSE) and their manufacture. To do so, we have benefited from the experience of four DSE projects. The main conclusions we came to are the following. First, DSE manufacturing consists in producing structured data, which shall, in a collaborative project, be validated against a schema. Second, schemas are meant to evolve over the lifespan of a DSE project, and may evolve even once some solid part of the data has been structured according to an initial data. Third, that schema evolution could be done collaboratively, and that the only collaborative way to define a schema was based on schema evolution.

Our first proposition (P. I Ch. 2) is a model of a collaborative DSE manufacturing process in which schemas are defined, and amended, in a collaborative, decentralised manner. The scenario we propose to explain how to collaboratively drive the evolution of a data structure is the following: from a situation in which there is a consensual data structure S , that is instantiated by all the editors (I_S represents the collective edition), one editor A , based on editorial considerations, defines an alternative data structure S' . The eAG/SeAG bidirectionalization ensures that, as soon as S' has been defined, a graph instantiating it, and retailing as much information from I_S as possible, is created, and synchronized with I_S , so that any update on one side is propagated on the other side.

Hence the need for a DSE-oriented data system to meet the following requirements: it must be schema-aware, that is, there must be a means to restrict the nature of the annotation written by an editor; it has to rest upon a data model (or an annotation formalism) expressive enough to enable the natural expression of some of the common patterns scholarly annotations are known to make use of (hierarchies of elements, attributes, distant relations between elements, multilayering, overlap and self overlap); it has to benefit from a tool-independent syntax, so that writing an annotation shall not demand the editor to use (and thus, for the DSE team, to develop and maintain) a dedicated interface; there must also be a means to define schema amendments, and to derive from a schema amendment an instance bidirectional transformation, so that an instance of the new schema shall be derived from the instance of the original schema as automatically as possible, and so that it shall be possible for an editor to implement any of the two schemas in an experimental approach, before taking the decision to opt for one schema or the other. Eventually – but this aspect was not considered in this

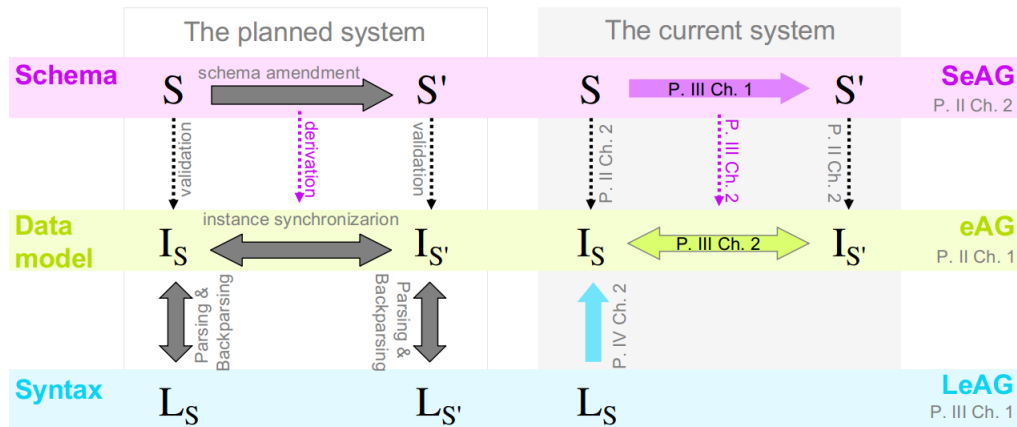


Figure 10.1: Summary of the editorial data system we intended to design (left), and the one we have defined (right).

work – editors implementing different schemas shall be made to discuss about the relevance of each, so that in the end one shall be chosen (instead of keeping two parallel schemas indefinitely). A corresponding such system is represented in Figure 10.1.

Our proposition revolves around three models: eAG, a cyclic annotation graph model that belongs to the class of stand-off markup models, and as such benefits from a high expressivity; SeAG, that is a schema language for eAG, based on the notion of simulation; LeAG, that is an inline markup syntax for eAG, that is, a tag-based syntax à la XML, but fit for the expression of multilayer annotation with distant relations and (self) overlap. As evidenced in Part II, Chapter 2, eAG validation can be guaranteed by construction: given a certain schema, it is possible to restrict the expression of valid-only eAGs, based on a matrix representation of both the schema and the graph. This offers the possibility to validate cyclic and multilayer annotation data on-the-fly, bypassing the traditional tradeoff between data model expressivity and data validation. We are not aware of any comparably ‘efficient’ validation mechanism for cyclic graph structured data.

We have also defined a schema-aware parsing algorithm for translating a LeAG into an eAG, given a SeAG. This parser, whose theoretical complexity is time-linear, permits to validate a LeAG: a LeAG is valid against a SeAG S iff the result of its parsing, given S , is defined. It results in the first validation mechanism running in linear time for the validation of cyclic data expressed in an inline markup syntax, to the best of our knowledge.

Eventually, in Part IV, we have defined, in an algebraic approach, several operators for amending schemas. We have also defined how to derive a bidirectional instance transformation from a schema transformation, that seems to be the first application of the Data Exchange problem to multistructured data and the first symmetric bidirectional transformations for (cyclic) graph-structured data.

The summary of the resulting system is provided in Figure 10.1.

10.10 A Word on the Adopted Methodology

This whole work has been conducted in two phases, with two different approaches. On the one side is the definition of the core problematic, that originates in editorial considerations, and of the editorial process meant as an answer to this problematic, from a non technical point of view. This part of the work has been undergone in direct collaboration with the editorial teams associated with this work, by means of meetings and exchange of information (e.g. the history of the abstract model of the DSE, or the whole evolution of the DTD supporting the edition, etc.).

A technical report, giving a summary of the interactions with the editorial teams, and proposing the first assessment of the problematic that we have discussed in this dissertation, together with the editorial process designed as a solution to this problematic, was sent to the editors, presented orally and discussed live, in order to get a validation of the general direction this PhD work was to follow, in December 2014. The conclusions of this technical report has been used as the basis for the introductory chapter of this dissertation.

On the other side is the definition of a data system making the editorial process defined as an answer to this problematic teams possible, from a technical point of view: definition of the data model, schema model and annotation syntax; definition of the validation mechanism; definition of the LeAG parser; definition of the SeAG and eAG transformations. This work has been conducted on a theoretical basis only, in order for as many of its underlying properties to be well-understood and established.

Implementing the whole resulting system was once an objective of this PhD work, mainly in order to conduct experiments with the end users (the editors) in terms of usability. Many aspects of the system indeed have to be tested: are the editors comfortable with the LeAG syntax? is it expressive enough? How should a SeAG schema be defined? modified? How to evaluate the editorial quality of the annotation graphs resulting from either the backward or the forward eAG transformations?

The lack of implementation is, as one may have guessed, mainly due to a corresponding lack of time to achieve it; it is particularly true since implementing only part of the whole system (e.g. the LeAG parser only, etc.) would not have enabled to experiment on the most interesting aspects, that regard the overall usability of the system.

Still, it has to be mentioned that the eAG/SeAG model, together with the SeAG operators for amending the schemas, have indeed been presented to the complete editorial team of the *Encyclopédie* project in a three-day workshop⁷. The model was, at the time, in a transitory state; for that reason, we did not set any proper user study on that occasion, which was more a means to share ideas and get feel how editors respond to such models. We got very enriching interactions, and an extra session was improvised for a dozen of the editors we met there, during which we studied, on a toy example, how to define a SeAG schema and how to compare two schemas.

⁷Journées d'étude ENCCRE, CIRM, Campus de Luminy, France, 5-9 octobre 2015.

10.11 Future Work and Perspectives

The system we have come up to, at the end of this PhD work, covers most of the aspects the initial problematic demanded to solve: in particular, we have proposed a schema-aware, multistructured data model with a very high expressivity while maintaining the time complexity of validation linear; we have defined a first solution to the problem of Data Exchange for cyclic annotation graphs. Yet, of course, this system can be complemented and improved in many aspects.

The first improvement would be to extend the eAG *Mod* transformation to the general case, where the schematic cells are either partially independent or ambiguous. The second improvement regards the temporal model of updates we have chosen here for the sake of simplicity, and that is not realistic.

We can also think of many ways to complement the propositions we formulate here. The first such complement, that is highlighted on Figure 10.1, would be to define a means to translate an eAG into a LeAG, and to make sure that this translation, together with the LeAG parser defined in Part III Chapter 2, behave like a bidirectional transformation.

The second complement would be to study how to check the well-formedness of a LeAG, an eAG, a SeAG efficiently. This could be part of a general reflection about how those models should be manipulated by the end user – which would be the right occasion for studying if well-formedness can be guaranteed by construction also.

The second complement may be inspired from the examples of complex SeAG transformations presented in Paragraph 10.7 page 294. Those examples show that a simple transformation like changing a single-occurrence SeAG pattern into a repeated pattern (called ‘cyclization’ above) demands a complex strategy from the user, if this one is to define schema amendments by means of the *Mod*, *Unite* and *Split* operators. One may wonder if a declarative approach, based on the algebra proposed here, should not be preferred.

The last complement would be to define a query language and a transformation language for eAG. Simulation, as we have seen, is the mechanism operated in SeAG for defining patterns that are then instantiated. It has been established that simulation can also be used for pattern matching in graph databases [186, 70]. An adaptation of such techniques to eAG is thus a track to follow.

Bibliography

Bibliography

- [1] The devonshire manuscript/general introduction. https://en.wikibooks.org/wiki/The_Devonshire_Manuscript/General_Introduction, Aug. 2017.
- [2] Travail collaboratif. http://fr.wikipedia.org/w/index.php?title=Travail_collaboratif&oldid=98902292, Aug. 2017.
- [3] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: from relations to semistructured data and XML*. Morgan Kaufmann, 2000.
- [4] L. Aceto, A. Ingolfsdottir, and J. Srba. The algorithmics of bisimilarity. *Advanced Topics in Bisimulation and Coinduction*, pages 100–172, 2012.
- [5] C. Allauzen and M. Mohri. *A unified construction of the Glushkov, Follow, and Antimirov automata*. Springer, 2006.
- [6] J. André and E. Pierazzo. Le codage en tei des brouillons de proust: vers l'édition numérique. *Genesis. Manuscrits-Recherche-Invention*, (36):155–161, 2013.
- [7] C. Andrews and C. North. Analyst's workspace: An embodied sensemaking environment for large, high-resolution displays. In *Visual Analytics Science and Technology (VAST), 2012 IEEE Conference on*, pages 123–131. IEEE, 2012.
- [8] M. Arenas and L. Libkin. Xml data exchange: consistency and query answering. *Journal of the ACM (JACM)*, 55(2):7, 2008.
- [9] M. Arenas, J. Pérez, J. Reutter, and C. Riveros. Inverting schema mappings: bridging the gap between theory and practice. *Proceedings of the VLDB Endowment*, 2(1):1018–1029, 2009.
- [10] M. Arenas, J. Pérez, J. Reutter, and C. Riveros. Composition and inversion of schema mappings. *ACM SIGMOD Record*, 38(3):17–28, 2010.
- [11] M. Assante, P. Pagano, L. Candela, F. De Faveri, and L. Lelii. An approach to virtual research environment user interfaces dynamic construction. In *Research and Advanced Technology for Digital Libraries*, pages 101–109. Springer, 2011.
- [12] G. Barabucci, A. Di Iorio, S. Peroni, F. Poggi, and F. Vitali. Annotations with earmark in practice: a fairy tale. In *Proceedings of the 1st International Workshop on Collaborative Annotations in Shared Environment: metadata, vocabularies and techniques in the Digital Humanities*, page 11. ACM, 2013.

- [13] P. Barceló, J. Pérez, and J. Reutter. Schema mappings and data exchange for graph databases. In *Proceedings of the 16th International Conference on Database Theory*, pages 189–200. ACM, 2013.
- [14] V. Barrellon and A. Guilbaud. Origami : première démonstration et perspectives de développement. *Claude simon, les vies de l'archive*, pages 211–224, 2014.
- [15] V. Barrellon, P.-E. Portier, S. Calabretto, and O. Ferret. Schema-aware extended annotation graphs. In *Proceedings of the 2016 ACM symposium on Document engineering*. ACM, 2016.
- [16] S. Bauman. Interchange vs. interoperability. In *Proceedings of Balisage: The Markup Conference 2011*, 2011.
- [17] S. Benda, J. Klímek, and M. Nečaský. Using schematron as schema language in conceptual modeling for xml. In *Proceedings of the Ninth Asia-Pacific Conference on Conceptual Modelling-Volume 143*, pages 31–40. Australian Computer Society, Inc., 2013.
- [18] P. A. Bernstein and S. Melnik. Model management 2.0: manipulating richer mappings. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1–12. ACM, 2007.
- [19] S. Bird, P. Buneman, and W.-C. Tan. Towards a query language for annotation graphs. *arXiv preprint cs/0007023*, 2000.
- [20] S. Bird, D. Day, J. Garofolo, J. Henderson, C. Laprun, and M. Liberman. Atlas: A flexible and extensible architecture for linguistic annotation. *arXiv preprint cs/0007022*, 2000.
- [21] S. Bird and M. Liberman. A formal framework for linguistic annotation. *Speech communication*, 33(1):23–60, 2001.
- [22] J. P. Birnholtz, T. A. Finholt, D. B. Horn, and S. J. Bae. Grounding needs: achieving common ground via lightweight chat in large, distributed, ad-hoc groups. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 21–30. ACM, 2005.
- [23] A. Bohannon, B. C. Pierce, and J. A. Vaughan. Relational lenses: a language for updatable views. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 338–347. ACM, 2006.
- [24] I. Boneva, A. Bonifati, and R. Ciucanu. Graph data exchange with target constraints. In *EDBT/ICDT Workshops*, pages 171–176, 2015.
- [25] R. Book, S. Even, S. Greibach, and G. Ott. Ambiguity in graphs and expressions. *Computers, IEEE Transactions on*, 100(2):149–153, 1971.
- [26] B. Bordalejo. The commedia project encoding system. *Dante Alighieri: The Commedia. A Digital Edition. Birmingham: Scholarly Digital Editions and Florence: Sismel*, 2010.

- [27] D. Botstein, J. M. Cherry, M. Ashburner, C. Ball, J. Blake, H. Butler, A. Davis, K. Dolinski, S. Dwight, J. Eppig, et al. Gene ontology: tool for the unification of biology. *Nat Genet*, 25(1):25–9, 2000.
- [28] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (xml) 1.0, 2008.
- [29] R. Bromme. Beyond one's own perspective: The psychology of cognitive interdisciplinarity. *Practicing interdisciplinarity*, pages 115–133, 2000.
- [30] A. Brüggemann-Klein. Regular expressions into finite automata. *Theoretical Computer Science*, 120(2):197–213, 1993.
- [31] A. Brüggemann-Klein. Compiler-construction tools and techniques for sgml parsers: Difficulties and solutions. In *Electronic Publishing: Origination, Dissemination and Design*. Citeseer, 1994.
- [32] A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Information and computation*, 140(2):229–253, 1998.
- [33] G. Brüning, K. Henzel, and D. Pravida. Multiple encoding in genetic editions: the case of "faust". *Journal of the TEI*, (4), 2013.
- [34] E. Bruno and E. Murisasco. Describing and querying hierarchical xml structures defined over the same textual data. In *Proceedings of the 2006 ACM symposium on Document engineering*, pages 147–154. ACM, 2006.
- [35] E. Bruno and E. Murisasco. Msxd: a model and a schema for concurrent structures defined over the same textual data. In *Database and Expert Systems Applications*, pages 172–181. Springer, 2006.
- [36] E. Bruno and E. Murisasco. An xml environment for multistructured textual documents. In *Digital Information Management, 2007. ICDIM'07. 2nd International Conference on*, volume 1, pages 230–235. IEEE, 2007.
- [37] P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Adding structure to unstructured data. In *Database Theory—ICDT'97*, pages 336–350. Springer, 1997.
- [38] P. Buneman, M. Fernandez, and D. Suciu. Unql: a query language and algebra for semistructured data based on structural recursion. *The VLDB Journal—The International Journal on Very Large Data Bases*, 9(1):76–110, 2000.
- [39] L. Burnard, S. Bauman, et al. *TEI P5: Guidelines for electronic text encoding and interchange*. TEI Consortium, 2008.
- [40] D. Buzzetti and J. McGann. Electronic textual editing: Critical editing in a digital horizon. *Text Encoding Initiative*, 2006.
- [41] S. Cassidy and J. Harrington. Multi-level annotation in the emu speech database management system. *Speech Communication*, 33(1):61–77, 2001.

- [42] H. A. Cayless. Rebooting tei pointers. *Journal of the Text Encoding Initiative*, (6), 2013.
- [43] B. Cerquiglini. Éloge de la variante. *Langages*, (69):25–35, 1983.
- [44] C. Chastain. Reference and context. In K. Gunderson, editor, *Language, Mind, and Knowledge*, volume 7, chapter 4, pages 194–231. University of Minnesota Press, 1975.
- [45] H. H. Clark. *Using language*, volume 4. Cambridge University Press Cambridge, 1996.
- [46] H. H. Clark and S. E. Brennan. Grounding in communication. *Perspectives on socially shared cognition*, 13(1991):127–149, 1991.
- [47] H. H. Clark and E. F. Schaefer. Contributing to discourse. *Cognitive science*, 13(2):259–294, 1989.
- [48] J. Clark. The design of relax ng. 2001.
- [49] J. Clark and M. Murata. {Relax NG} specification. 2001.
- [50] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: <http://www.grappa.univ-lille3.fr/tata>, 2007. release October, 12th 2007.
- [51] G. O. Consortium et al. The gene ontology project in 2008. *Nucleic acids research*, 36(suppl 1):D440–D444, 2008.
- [52] G. Convertino, H. M. Mentis, M. B. Rosson, J. M. Carroll, A. Slavkovic, and C. H. Ganoe. Articulating common ground in cooperative work: content and process. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1637–1646. ACM, 2008.
- [53] G. Convertino, H. M. Mentis, M. B. Rosson, A. Slavkovic, and J. M. Carroll. Supporting content and process common ground in computer-supported teamwork. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2339–2348. ACM, 2009.
- [54] G. Convertino, H. M. Mentis, A. Y. Ting, M. B. Rosson, and J. M. Carroll. How does common ground increase? In *Proceedings of the 2007 international ACM conference on Supporting group work*, pages 225–228. ACM, 2007.
- [55] D. Cristea, N. Ide, L. Romary, et al. Marking-up multiple views of a text: Discourse and reference. In *Proceedings of the First International Conference on Language Resources and Evaluation*, 1998.
- [56] C. Crompton, D. Powell, A. Arbuckle, R. Siemens, M. Shirley, et al. Building a social edition of the devonshire manuscript. 2014.

- [57] K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, and J. F. Terwilliger. Bidirectional transformations: A cross-discipline perspective. In *Theory and Practice of Model Transformations*, pages 260–283. Springer, 2009.
- [58] J. de la Rosa and D. M. Brown. Curation, management, and analysis of highly connected data in the humanities. *Digital Humanities*, pages 38–40, 2014.
- [59] A. Di Iorio, S. Peroni, and F. Vitali. Using semantic web technologies for analysis and validation of structural markup. *International Journal of Web Engineering and Technology*, 6(4):375–398, 2011.
- [60] W. Dillen. Sequentiality in genetic digital scholarly editions. models for encoding the dynamics of the writing process. *Digital Humanities*, pages 174–175, 2016.
- [61] P. D’Iorio and M. Barbera. Scholarsource: A digital infrastructure for the humanities. *Switching Codes. Thinking through New Technology in the Humanities and the Arts*, pages 61–87, 2011.
- [62] A. Dix. *Human computer interaction*. Pearson Education, 2004.
- [63] A. Donnellon, B. Gray, and M. G. Bougon. Communication, meaning, and organized action. *Administrative Science Quarterly*, pages 43–55, 1986.
- [64] S. Dord-Crouslé and E. Morlock-Gerstenkorn. Le " modèle abstrait " du corpus bouvard: première approche. In *journée d’étude " Constitution et exploitation de corpus issus de manuscrits-Lectures, écritures et nouvelles approches en recherche documentaire " organisée par Cécile Meynard et Thomas Lebarbé*, 2009.
- [65] P. D’Iorio. L’île des savoirs choisis. d’hypernietzsche à scholarsource: pour une infrastructure de recherche sur le web. *Recherches & Travaux*, (72):279–301, 2008.
- [66] S. Ehlinger. Les représentations partagées au sein des organisations: entre mythe et réalité. *Actes du 8ème congrès de l’AIMS*, 1998.
- [67] H. Ehrig, C. Ermel, F. Hermann, and U. Prange. On-the-fly construction, correctness and completeness of model transformations based on triple graph grammars. In *MoDELS*, volume 9, pages 241–255. Springer, 2009.
- [68] O. Eide. Sequence, tree and graph at the tip of your java classes. *Digital Humanities*, pages 152–154, 2014.
- [69] R. Fagin. Inverting schema mappings. *ACM Transactions on Database Systems (TODS)*, 32(4):25, 2007.
- [70] W. Fan, J. Li, S. Ma, N. Tang, and Y. Wu. Adding regular expressions to graph reachability and pattern queries. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 39–50. IEEE, 2011.
- [71] F. Ferlin. D’alembert et l’optique: l’encyclopédie comme banc d’essai de recherches originales. *Recherches sur Diderot et sur l’Encyclopédie*, (43):127–144, 2008.

- [72] O. Ferret. Les “réflexions philosophiques” dans les éloges académiques de d’alembert : le cas de l’Éloge de bossuet. *Bollettino di storia delle scienze matematiche*, vol. XXVIII(fasc. 2):255–272, 2008.
- [73] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, and A. Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 29(3):17, 2007.
- [74] G. Franzini, M. Terras, and S. Mahony. A catalogue of digital editions. *Digital Scholarly Editing*, page 161.
- [75] B. Frellesvig, S. W. Horn, K. L. Russell, and P. Sells. The oxford corpus of old japanese. *Electronic publication, accessible at <http://vsarpj.orinst.ox.ac.uk/corpus>*, 2014.
- [76] J. P. Fry, D. P. Smith, and R. W. Taylor. An approach to stored data definition and translation. In *Proceedings of 1972 ACM-SIGFIDET workshop on Data description, access and control*, pages 13–55. ACM, 1972.
- [77] H. W. Gabler. Theorizing the digital scholarly edition. *Literature Compass*, 7(2):43–56, 2010.
- [78] S. Gao, C. Sperberg-McQueen, and H. S. Thompson. W3c xml schema definition language (xsd) 1.1 part 1: Structures. w3c recommendation 5 april 2012. *World Wide Web Consortium*. <http://www.w3.org/TR/xmlschema11-1/>. Accessed, 30, 2013.
- [79] W. Gelade, W. Martens, and F. Neven. Optimizing schema languages for xml: Numerical constraints and interleaving. In *Database Theory-ICDT 2007*, pages 269–283. Springer, 2007.
- [80] W. Gelade and F. Neven. Succinctness of the complement and intersection of regular expressions. *ACM Transactions on Computational Logic (TOCL)*, 13(1):4, 2012.
- [81] R. Gentilini, C. Piazza, and A. Policriti. From bisimulation to simulation: Coarsest partition problems. *Journal of Automated Reasoning*, 31(1):73–103, 2003.
- [82] E. Geoffrois, C. Barras, S. Bird, and Z. Wu. Transcribing with annotation graphs. In *LREC*, 2000.
- [83] D. A. Gioia and H. P. Sims. *The thinking organization*. San Francisco: Jossey-Bass, 1986.
- [84] V. M. Glushkov. The abstract theory of automata. *Russian Mathematical Surveys*, 16(5):1–53, 1961.
- [85] C. F. Goldfarb and Y. Rubinsky. *The SGML handbook*. Oxford University Press, 1990.

- [86] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. Technical report, Stanford, 1997.
- [87] J. Grudin and S. Poltrock. *Computer Supported Cooperative Work*. The Interaction Design Foundation, 2013.
- [88] D. Grune and C. J. Jacobs. A programmer-friendly ll (1) parser generator. *Software: Practice and Experience*, 18(1):29–38, 1988.
- [89] Y.-S. Han and D. Wood. Generalizations of 1-deterministic regular languages. *Information and Computation*, 206(9):1117–1125, 2008.
- [90] F. Henri and K. Lundgren-Cayrol. *Apprentissage collaboratif à distance*. Presses de l’Université du Québec, 2001.
- [91] F. Hermann, H. Ehrig, F. Orejas, K. Czarnecki, Z. Diskin, and Y. Xiong. Correctness of model synchronization based on triple graph grammars. In *MoDELS*, volume 11, pages 668–682. Springer, 2011.
- [92] F. Hermann, N. Rummel, and H. Spada. Solving the case together: The challenge of net-based interdisciplinary collaboration. *European perspectives on computer-supported collaborative learning: Universiteit Maastricht*, 2001.
- [93] S. Hidaka, Z. Hu, K. Inaba, H. Kato, K. Matsuda, and K. Nakano. Groundtram version 0.9. 3a user manual. 2008.
- [94] S. Hidaka, Z. Hu, K. Inaba, H. Kato, and K. Nakano. Groundtram: An integrated framework for developing well-behaved bidirectional model transformations. In *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*, pages 480–483. IEEE, 2011.
- [95] M. Hilbert, A. Witt, and O. Schonefeld. Making concur work. In *In Extreme Markup Languages*, 2005.
- [96] J. A. Hill, B. E. Smith, P. G. Papoulias, and P. C. Andrews. Proteomecommons.org collaborative annotation and project management resource integrated with the tranche repository. *Journal of proteome research*, 9(6):2809–2811, 2010.
- [97] C. Huitfeldt and C. Sperberg-McQueen. Texmecs: An experimental markup meta-language for complex documents. URL <http://www.hit.uib.no/clus/mlcd/papers/termecs.html>, 2001.
- [98] N. Ide, L. Romary, and E. de la Clergerie. International standard for a linguistic annotation framework. In *Proceedings of the HLT-NAACL 2003 workshop on Software engineering and architecture of language technology systems- Volume 8*, pages 25–30. Association for Computational Linguistics, 2003.
- [99] L. Ilie and S. Yu. Follow automata. *Information and computation*, 186(1):140–162, 2003.
- [100] E. A. Isaacs and H. H. Clark. References in conversation between experts and novices. *Journal of Experimental Psychology: General*, 116(1):26, 1987.

- [101] H. Jagadish, L. V. Lakshmanan, M. Scannapieco, D. Srivastava, and N. Watwattana. Colorful xml: one hierarchy isn't enough. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 251–262. ACM, 2004.
- [102] F. Jannidis and J. Flanders. A concept of data modeling for the humanities. *Digital Humanities*, pages 237–39, 2013.
- [103] I. R. Johnson and A. Osmakov. Rebuilding digital harlem for sustainability and change. *Digital Humanities*, pages 812–813, 2016.
- [104] P. A. Kirschner, P. J. Beers, H. Boshuizen, and W. H. Gijselaers. Coercing shared knowledge in collaborative learning environments. *Computers in human behavior*, 24(2):403–420, 2008.
- [105] M. Klein, D. Fensel, F. Van Harmelen, and I. Horrocks. The relation between ontologies and xml schemas. *Electronic Trans. on Artificial Intelligence*, pages 128–145, 2001.
- [106] M.-J. Kline and S. H. Perdue. *A guide to documentary editing*. Johns Hopkins University Press, 1998.
- [107] H. Knublauch and A. Ryman. Shapes constraint language (shacl). *W3C First Public Working Draft*, 8:W3C, 2015.
- [108] P. G. Kolaitis. Schema mappings, data exchange, and metadata management. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 61–75. ACM, 2005.
- [109] T. Koschmann and C. D. LeBaron. Reconsidering common ground. In *ECSCW 2003*, pages 81–98. Springer, 2003.
- [110] J. Le Rond d'Alembert. Discours préliminaire. *Encyclopédie, ou Dictionnaire raisonné des sciences, des arts et des métiers*. Paris: Briasson, David, Le Breton & Durand, 1751.
- [111] J.-L. Lebrave. La critique génétique: une discipline nouvelle ou un avatar moderne de la philologie? *Genesis (Manuscrits-Recherche-Invention)*, 1(1):33–72, 1992.
- [112] D. Lee, M. Mani, and M. Murata. Reasoning about xml schema languages using formal language theory. Technical report, Citeseer, 2000.
- [113] V. Y. Lum, N. C. Shu, and B. C. Housel. A general methodology for data conversion and restructuring. *IBM Journal of research and development*, 20(5):483–497, 1976.
- [114] K. Mäkitalo, P. Häkkinen, P. Leinonen, and S. Järvelä. Mechanisms of common ground in case-based web discussions in teacher education. *The Internet and Higher Education*, 5(3):247–265, 2002.

- [115] M. Mani. Keeping chess alive-do we need 1-unambiguous content models. *Extreme Markup Languages, Montreal, Canada*, 2001.
- [116] W. Martens, F. Neven, T. Schwentick, and G. J. Bex. Expressiveness and complexity of xml schema. *ACM Transactions on Database Systems (TODS)*, 31(3):770–813, 2006.
- [117] P. McBrien and A. Poulouvassilis. Schema evolution in heterogeneous database architectures, a schema transformation approach. In *Advanced Information Systems Engineering*, pages 484–499. Springer, 2002.
- [118] J. C. McCarthy, V. C. Miles, and A. F. Monk. An experimental study of common ground in text-based communication. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 209–215. ACM, 1991.
- [119] W. McCarty. Modeling: a study in words and meanings. *A companion to digital humanities*, 42:254, 2004.
- [120] W. McCarty and M. Deegan. *Collaborative research in the digital humanities*. Routledge, 2016.
- [121] J. McGann. The rationale of hypertext. In *Radiant Textuality*, pages 53–74. Springer, 2001.
- [122] M. J. McGuffin et al. A comparison of hyperstructures: Zzstructures, mspaces, and polyarchies. In *Proceedings of the fifteenth ACM conference on Hypertext and hypermedia*, pages 153–162. ACM, 2004.
- [123] R. Milner. *Communication and Concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.
- [124] N. Miyake. Computer supported collaborative learning. *The SAGE handbook of e-learning research*, pages 248–265, 2007.
- [125] E. V. Munson. Collaborative authoring requires advanced change management. In *DChanges*, 2013.
- [126] M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of xml schema languages using formal language theory. *ACM Transactions on Internet Technology (TOIT)*, 5(4):660–704, 2005.
- [127] N. F. Noy and M. Klein. Ontology evolution: Not the same as schema evolution. *Knowledge and information systems*, 6(4):428–440, 2004.
- [128] G. Ott and N. H. Feinstein. Design of sequential machines from their regular expressions. *Journal of the ACM (JACM)*, 8(4):585–600, 1961.
- [129] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

- [130] F. Papazian, R. Bossy, and C. Nédellec. Alvisae: a collaborative web text annotation editor for knowledge acquisition. In *Proceedings of the Sixth Linguistic Annotation Workshop*, pages 149–152. Association for Computational Linguistics, 2012.
- [131] T. M. Paulus. Online but off-topic: Negotiating common ground in small learning groups. *Instructional Science*, 37(3):227–245, 2009.
- [132] S. Peroni. Markup beyond the trees. In *Semantic Web Technologies and Legal Scholarly Publishing*, pages 45–93. Springer, 2014.
- [133] S. Peroni, F. Poggi, and F. Vitali. Tracking changes through earmark: a theoretical perspective and an implementation. In *DChanges*, 2013.
- [134] M. Pichat and C. Solnon. *Théorie des langages*, 2005.
- [135] E. Pierazzo. Digital documentary editions and the others. *Scholarly Editing*, 35:1–23, 2014.
- [136] E. Pierazzo. *Digital Scholarly Editing: Theories, Models and Methods*. Ashgate Publishing, Ltd., 2015.
- [137] B. C. Pierce. The weird world of bi-directional programming. *Lecture Notes in Computer Science*, 3924:342, 2006.
- [138] W. Piez. *Data Modeling for the Humanities: Three Questions and One Experiment*. Workshop on Knowledge Organization and Data Modeling in the Humanities, Brown University, March 2013. [Accessed: 2016 06 20].
- [139] W. Piez. Tei in lmn1: Implications for modeling. *Journal of the Text Encoding Initiative*, (8), 2015.
- [140] M. Poesio, R. Stuckardt, and Y. E. Versley. *Anaphora Resolution: Algorithms, Resources, and Applications*. Springer-Verlag Berlin Heidelberg, 2016.
- [141] P.-É. Portier, N. Chatti, S. Calabretto, E. Egyed-Zsigmond, and J.-M. Pinon. Modeling, encoding and querying multi-structured documents. *Information Processing & Management*, 48(5):931–955, 2012.
- [142] K. M. Price. Electronic scholarly editions. *A companion to digital literary studies*, pages 434–450, 2007.
- [143] E. Prud’hommeaux, J. E. Labra Gayo, and H. Solbrig. Shape expressions: an rdf validation and transformation language. In *Proceedings of the 10th International Conference on Semantic Systems*, pages 32–40. ACM, 2014.
- [144] F. Ranzato and F. Tapparo. An efficient simulation algorithm based on abstract interpretation. *Information and Computation*, 208(1):1–22, 2010.
- [145] A. H. Renear, E. Mylonas, and D. Durand. Refining our notion of what text really is: The problem of overlapping hierarchies. 1993.

- [146] D. Reynolds, C. Thompson, J. Mukerji, and D. Coleman. An assessment of rdf/owl modelling. *Digital Media Systems Laboratory, HP Laboratories Bristol*, 28, 2005.
- [147] P. Robinson. *Towards a theory of digital editions*, volume 10. Variants, 2013.
- [148] P. Robinson. Social editions, social editing, social texts. *Digital Studies/Le champ numérique*, 2016.
- [149] M. Roggenbach and M. Majster-Cederbaum. Towards a unified view of bisimulation: a comparative study. *Theoretical Computer Science*, 238(1):81–130, 2000.
- [150] R. e. a. Sanderson. Open annotation data model. *W3C community draft*, 2013.
- [151] D. Schmidt. The inadequacy of embedded markup for cultural heritage texts. *Literary and Linguistic Computing*, 25(3):337–356, 2010.
- [152] O. Schonefeld and A. Witt. Towards validation of concurrent markup. In *Proceedings of the extreme markup languages*, 2006.
- [153] A. Schürr and F. Klar. 15 years of triple graph grammars. In *Icgt*, pages 411–425. Springer, 2008.
- [154] P. Scifleet and S. Williams. Understanding documentary practice: lessons learnt from the text encoding initiative. *Research and Advanced Technology for Digital Libraries*, pages 272–283, 2011.
- [155] P. L. Shillingsburg. *Scholarly editing in the computer age: Theory and practice*. University of Michigan Press, 1996.
- [156] F. M. Shipman and C. C. Marshall. *Formality Considered Harmful: Experiences, Emerging Themes, and Directions*. University of Colorado, Boulder, Department of Computer Science, 1993.
- [157] L. Siemens, R. Cunningham, W. Duff, and C. Warwick. “more minds are brought to bear on a problem”: Methods of interaction and collaboration within digital humanities research teams. *Digital Studies/Le champ numérique*, 2(2), 2010.
- [158] R. Siemens, M. Timney, C. Leitch, C. Koolen, and A. Garnett. Toward modeling the social edition: An approach to understanding the electronic scholarly edition in the context of new and emerging social media. *Literary and Linguistic Computing*, 27(4):445–461, 2012.
- [159] E. Sirin. Data validation with owl integrity constraints. In *Web Reasoning and Rule Systems*, pages 18–22. Springer, 2010.
- [160] C. M. Sperberg-McQueen. Rabbit/duck grammars: a validation method for overlapping structures. In *Extreme Markup Languages*, 2006.

- [161] C. M. Sperberg-McQueen, L. Burnard, et al. *Guidelines for electronic text encoding and interchange*, volume 1. Text Encoding Initiative Chicago and Oxford, 1994.
- [162] C. M. Sperberg-McQueen and C. Huitfeldt. Goddag: A data structure for overlapping hierarchies. In *Digital documents: Systems and principles*, pages 139–160. Springer, 2000.
- [163] M. Sperberg-McQueen. Closing keynote session. Workshop on Knowledge Organization and Data Modeling in the Humanities, 2012.
- [164] M. C. Sperberg McQueen. Taking modeling seriously: A hands-on approach to alloy. *Digital Humanities*, pages 27–28, 2013.
- [165] G. Stahl. *Group cognition*, volume 106. Citeseer, 2006.
- [166] R. Stalnaker. Assertion. *Syntax and Semantics*, 9:315–332, 1978.
- [167] S. Staworko, I. Boneva, J. E. Labra Gayo, S. Hym, E. G. Prud’hommeaux, and H. Solbrig. Complexity and expressiveness of shex for rdf. In *LIPICs-Leibniz International Proceedings in Informatics*, volume 31. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.
- [168] Stendhal, C. Meynard, H. de Jacquelot, and M.-R. Corredor. *Journaux et papiers*. Ellug, 2013.
- [169] L. Stojanovic, A. Maedche, B. Motik, and N. Stojanovic. User-driven ontology evolution management. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 285–300. Springer, 2002.
- [170] M. Stührenberg and C. Wurm. Refining the taxonomy of xml schema languages. a new approach for categorizing xml schema languages in terms of processing complexity. In *Proceedings of Balisage: The Markup Conference*, volume 5, 2010.
- [171] D. Suciu. Semistructured data and xml. In *Information organization and databases*, pages 9–30. Springer, 2000.
- [172] J. Tao, E. Sirin, J. Bao, and D. L. McGuinness. Integrity constraints in owl. In *AAAI*, 2010.
- [173] J. Tennison. Validating xml with schematron. In *Beginning XSLT*, pages 626–660. Springer, 2004.
- [174] J. Tennison. Creole: Validating overlapping markup. In *Proceedings of XTech*, 2007.
- [175] J. Tennison and W. Piez. The layered markup and annotation language (lxml). In *Extreme Markup Languages*, 2002.
- [176] C. Thao and E. V. Munson. Using versioned tree data structure, change detection and node identity for three-way xml merging. In *Proceedings of the 10th ACM symposium on Document engineering*, pages 77–86. ACM, 2010.

- [177] K. Tomasek and S. Bauman. Problems in modeling transactions. *Digital Humanities*, pages 385–386, 2014.
- [178] G. Tummarello, C. Morbidoni, and E. Pierazzo. Toward textual encoding based on rdf. In *ELPUB*, 2005.
- [179] E. S. Veinott, J. Olson, G. M. Olson, and X. Fu. Video helps remote work: Speakers who need to negotiate common ground benefit from seeing each other. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 302–309. ACM, 1999.
- [180] M. A. Walker. Clark’s *Using Language*, (review). *Computational linguistics*, 23(4):625–628, 1997.
- [181] B. W. Watson. *Taxonomies and toolkits of regular language algorithms*. Eindhoven University of Technology The Netherlands, 1995.
- [182] K. E. Weick. *The social psychology of organizing*. 1979.
- [183] A. Widlöcher and Y. Mathet. The glozz platform: A corpus annotation and mining tool. In *Proceedings of the 2012 ACM symposium on Document engineering*, pages 171–180. ACM, 2012.
- [184] A. Witt. Different views on markup. *Text, Speech and Language Technology*, page 1, 2010.
- [185] D. Wood. Standard generalized markup language: Mathematical and philosophical issues. In *Computer Science Today*, pages 344–365. Springer, 1995.
- [186] P. T. Wood. Query languages for graph databases. *ACM SIGMOD Record*, 41(1):50–60, 2012.
- [187] F. Yang and S. Hidaka. Bidirectional transformation on ordered graphs. Technical report, Technical Report 2015-08, GRACE Center, 2015., 2014.
- [188] L. Zamboulis. Xml data integration by graph restructuring. In *Key Technologies for Data Management*, pages 57–71. Springer, 2004.
- [189] H. Zhu, R. Kraut, and A. Kittur. Organizing without formal organization: group identification, goal setting and social modeling in directing online production. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 935–944. ACM, 2012.

Appendix

Chapter 11

eAG Example: Multiple Chronology-based Annotation.

Prenons pour base d'illustration la photographie suivante, qui représente le folio 67r du manuscrit "de Voynich" conservé à la bibliothèque de Yale¹. Ce manuscrit présente pour nous l'intérêt d'être indéchiffré, et donc illisible à ce jour même pour les plus érudits : nous pourrions donc imaginer, à titre d'exemple, toutes les hypothèses de lecture sans pouvoir être contredits, ce qui est un luxe dans notre situation de profane égarés dans le monde des lettres... Voici le folio :

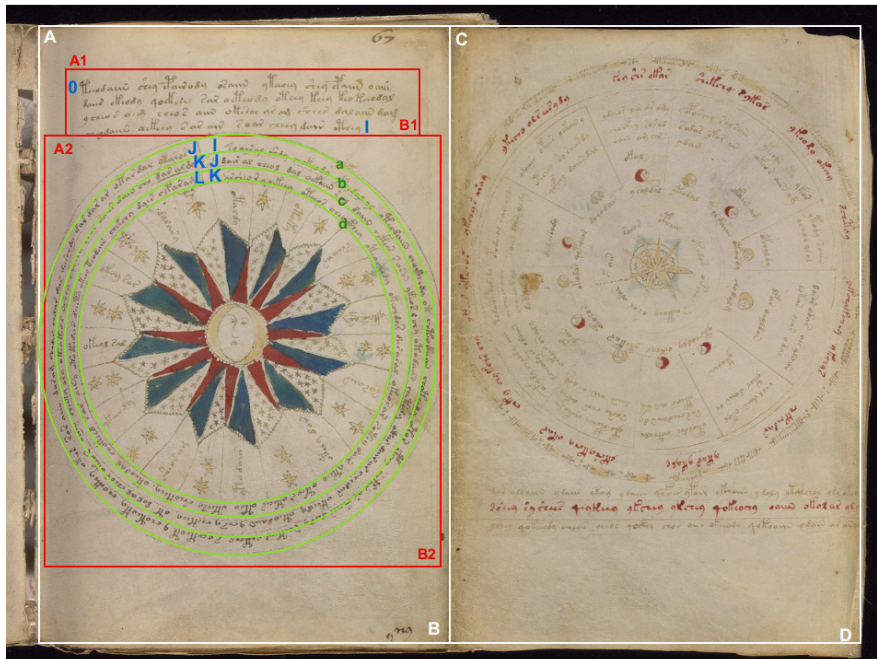


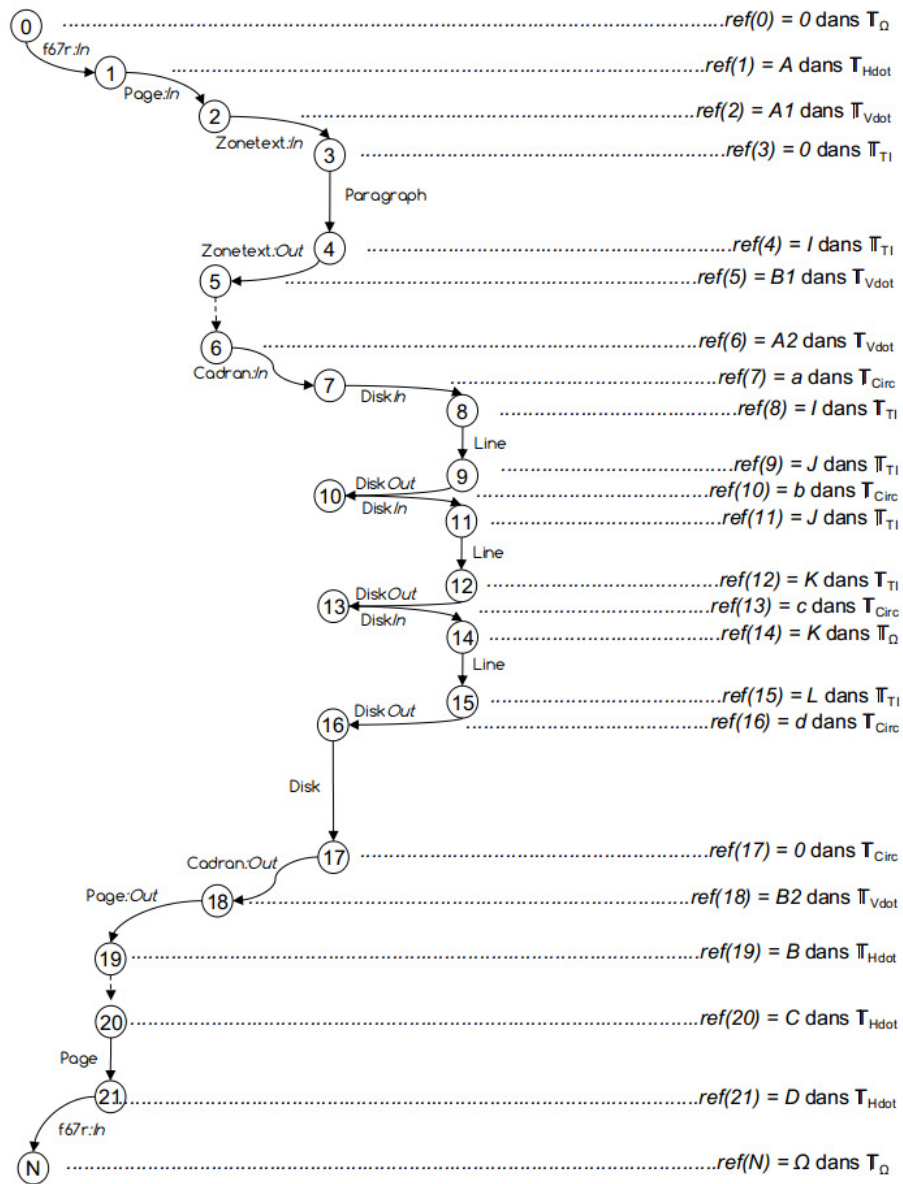
¹Voynich Manuscript Cipher Manuscript, Beinecke MS 408 67r, Beinecke Rare Book and Manuscript Library, Yale University.

Nous nous munissons de cinq chronomètres pour baliser cette image :

1. Le premier, nommé *Hdot*, tel que $\mathbb{T}_{Vdot} = \{(x, y) : x, y \in \mathbb{R}\}$ et $(x, y) <_{Hdot} (x', y') \Leftrightarrow y < y' \vee (y = y' \wedge x < x')$. Il identifie des nœuds avec une prépondérance des y dans la relation d'ordre (l'axe des y étant pris comme l'axe de horizontal orienté de gauche à droite).
2. Le second, nommé *Vdot*, $\mathbb{T}_{Vdot} = \{(x, y) : x, y \in \mathbb{R}\}$ et $(x, y) <_{Vdot} (x', y') \Leftrightarrow x < x' \vee (x = x' \wedge y < y')$. Il identifie des nœuds avec une prépondérance des x dans la relation d'ordre (l'axe des x étant pris comme l'axe de vertical orienté de haut en bas).
3. Le troisième, nommé *Circ*, tel que $\mathbb{T}_{Circ} = \{x \in \mathbb{R}^+\}$ et $x <_{Circ} x' \Leftrightarrow x > x'$. Il identifie des cercles concentriques, dont le centre est le centre de la zone décrite, par leur rayon. L'ordre est calculé selon l'inverse du rayon.
4. Le quatrième, nommé *TI*, tel que $\mathbb{T}_{TI} = \{x \in \mathbb{N}\}$ et $x <_{TI} x' \Leftrightarrow x < x'$. Il identifie une position entre deux caractères dans une chaîne de caractères.
5. Un dernier, nommé Ω , tel que $\mathbb{T}_{\Omega} = \{0; \Omega\}$ et $x <_{TextIndex} x' \Leftrightarrow x = 0 \wedge x' = \Omega$. Il ne sert qu'à indexer la racine et la feuille.

Nous proposons à titre illustratif l'annotation suivante du folio 67r :





On identifie dans cet exemple sept niveaux hiérarchiques. Nous les décrivons ici, ainsi que les chronologies auxquelles ils font référence :

1. $N_0 = \{0; N\}$ associé à $T_0 = [0; \Omega]$ sur T_Ω .
2. $N_1 = \{1; 19; 20; 21\}$ associé à $T_1 = [A; B] \cup [C; D]$ sur T_{Hdot} , avec $A = (0, 0)$, $B = (N, M)$, $C = (0, M + 1)$, $D = (N, 2M)$.

3. $N_2 = \{2; 5; 6; 18\}$ associé à $T_2 = [A1; B1] \cup [A2; B2]$ sur \mathbb{T}_{Vdot} , avec $A1 = (X + x, Y)$, $B1 = (X + x + x', Y + y')$, $A2 = (X, Y + y' + 1)$, $B2 = (X + x + x' + x'', Y + y' + y'')$.
4. $N_3 = \{3; 4\}$ associé à $T_3 = [0; I]$ sur \mathbb{T}_{TI} .
5. $N_4 = \{7; 10; 13; 16; 17\}$ associé à $T_4 = [0; a]$ sur \mathbb{T}_{Circ} .
6. $N_5 = \{8; 9\}$ associé à $T_5 = [I; J]$ sur \mathbb{T}_{TI} .
7. $N_6 = \{11; 12\}$ associé à $T_6 = [J; K]$ sur \mathbb{T}_{TI} .
8. $N_7 = \{14; 15\}$ associé à $T_7 = [K; L]$ sur \mathbb{T}_{TI} .

Outre la lecture hiérarchique particulière que nous proposons de la page, qui est bien rendue par cette annotation, nous voulons insister sur quelques points qui nous paraissent intéressants :

- Notre modèle permet une description locale, une adaptation du référencement à la nature des contenus : par exemple, on définit un ordre concentrique sur \mathbb{T}_{Circ} pour annoter le cadran représenté en bas de la page de gauche du folio. Sur cet exemple, cela permet à l'éditeur de suggérer une lecture du texte compris dans les anneaux concentriques de l'extérieur vers l'intérieur (l'inverse aurait été possible). Plus généralement, cela offre la possibilité de traduire dans l'annotation des séquences non triviales spatialement.
- Cela permet également une adaptation du sens de lecture en fonction de l'échelle où on se situe : à ce titre, on propose ici un découpage du folio 67r en deux pages côte à côte, mais on définit un ordre de lecture vertical au sein de chaque page.
- En termes de comparaison de références de nœuds (et donc de localisation des items au sein du corpus) :
 - *Entre des nœuds faisant référence au même ensemble de référencement, mais n'appartenant pas au même niveau hiérarchique* : on peut par exemple comparer la position dans le corpus des portions encadrées respectivement par les nœuds (3, 4) et (14, 15). On peut conclure à l'antériorité référentielle du premier par rapport à l'autre, du fait que malgré la non appartenance au même niveau hiérarchique, comme aucun label de $\overline{\mathcal{L}}_{LinkTo} \cup \text{stuff}(\overline{\mathcal{L}}_{LinkTo}, \text{Stuff})$ n'appartient au chemin liant 4 à 13, la cohérence des ordres référentiels et structurels doit être respectée ;
 - *Entre des nœuds ne faisant pas référence au même ensemble de référencement*. Dans ce cas, ce n'est pas la valeur de la référence qui permet de conclure mais l'inclusion d'une chronologie dans relative à un nœud dans une chronologie relative à l'autre nœud, ou qui contient l'autre nœud (etc.). Cela permet de dire que dans notre transcription, le texte identifié par les nœuds 8 et 9 est contenu dans un disque délimité par les cercles représentés par les nœuds 7 et 10, car les premiers font référence à $T_5 \subset T_4$, et partant que $ref(7) < ref(8) <_{TI} ref(9) < ref(10)$.

Chapter 12

Tree-Automata vs. Simulation-based Validation

Nous voulons ici étudier le rapport entre la validation par simulation que l'on propose, pour des documents structurés en graphes cycliques, enracinés et enfeuillés, et la notion de validation mise en œuvre dans le cadre restreint de XML. Pour cela, nous nous référons au travail de Murata, Lee, Mani et al. ([126], [112]).

La validation de documents XML. Makoto Murata est l'un des deux fondateurs du langage de validation XML RELAX NG [49] avec James Clark. Dans l'article [126], Murata et al. s'appuient sur la notion de Tree Automata [50], dans le sens suivant :

- un document XML peut être modélisé par un arbre ;
- un schéma XML¹ est un Tree Automaton, c'est-à-dire un automate dont les résultats d'exécution sont des arbres ;
- un document XML est donc validé par un schéma si et seulement si le document correspond à une exécution possible du schéma.

Notre démarche est différente. Selon notre approche, un schéma est vu comme un automate validant des chaînes de symboles (possédant en outre un seul nœud d'entrée et un unique nœud de sortie), et un document est également vu comme un automate de la même nature. C'est-à-dire que dans notre cas, le document n'est pas un résultat d'exécution d'un automate, et dans le sens inverse, le schéma n'est pas un automate dont les résultats d'exécution sont des objets possédant la même structure que les documents.

La validation fondée sur la notion de Tree Automata exploite une relation appelée "interprétation" entre un document (un arbre) et un schéma. Nous la définissons ici formellement.

¹Nous désignons par ce terme un schéma rédigé dans n'importe quel formalisme : cela ne désigne pas spécifiquement les XML Schema définis par le W3C Consortium [78], mais aussi bien un DTD, un fichier RELAX NG. Dans l'esprit de [126], nous mettons à part Schematron (cf. [173]), qui précise moins les contraintes structurelles que les contraintes d'intégrité des documents.

Definition 12.1 : Grammaire d'arbres / Tree Automata On appelle un *Tree Automaton* un quadruplet $TA = (N, T, S, P)$ où :

- N est l'ensemble des symboles non terminaux ;
- T est l'ensemble des symboles terminaux ;
- $St \subseteq N$ est l'ensemble des symboles initiaux ;
- P est l'ensemble des règles de production des arbres, sous la forme $r = X \rightarrow aR$, où $r \in P$, $X \in N$, $a \in T$ et R est une expression régulière sur N .

Notation. A la suite de [126], nous notons $v(v_1, v_2, v_3)$ le fait que dans un arbre donné, le nœud v est le père de v_1, v_2 et v_3 .

Definition 12.2 : Interprétation Soit un arbre $t = (V, E)$ et $TA = (T, N, St, P)$. On dit qu'il existe une *interprétation* de t sur TA si et seulement si $\exists I : V \rightarrow N$ telle que :

1. $I(\text{root}(t)) \in St$;
2. $\forall v(v_1, \dots, v_N) \subset t, \exists r \in P$;
 - $r = X \rightarrow aR$;
 - $I(r) = X$;
 - $\text{label}(v) = a$;
 - $I(v_1) \cdot \dots \cdot I(v_N) \in \text{lang}(R)$.

On note l'existence d'une interprétation I de t sur TA : $\exists I(t)/TA$.

La condition de validation d'un arbre t par un Tree Automaton TA est l'existence de $I(t)/TA$. Notre notion de validation repose pour sa part sur la relation de simulation.

Nous démontrons ici que, malgré les différences entre les deux visions de la validation, la condition de validation par interprétation est représentable au moyen de notre condition de validation par simulation. Nous développons les notions nécessaires pour formuler clairement cette proposition et la démontrer. Le déroulement de cette démonstration est le suivant :

- Nous rappelons comment représenter un arbre t au moyen d'un AGm G .
- Nous définissons ensuite une dérivation d'un schéma S (dont on ne cherche pas à prouver qu'elle vérifie strictement notre modèle de schéma²) à partir d'un Tree Automata TA .
- Nous prouvons enfin que s'il existe une interprétation de t par TA , alors $S \leftrightarrow G$.

²Cette approximation n'est pas gênante : nous voulons comparer ici la relation d'interprétation et celle de simulation dans le contexte de la validation, pas dériver à proprement parler un schéma selon notre Definition depuis un Tree Automaton.

Commentaire. On peut remarquer deux choses à ce point :

1. la validation par un schéma RELAX NG ne se réduit pas à l'existence d'une interprétation : RELAX NG propose deux extensions aux grammaires régulières d'arbres, nommément des contraintes attribut-élément et l'entrelacement.
2. nous confrontons seulement les relations de simulation et d'interprétation, et non la simulation *typée* et l'interprétation. La raison en est que l'attribution cohérente de types aux nœuds de l'annotation graph dérivé d'un arbre et aux nœuds du schéma dérivé du Tree Automata présente un surcroît de difficultés qui rendrait illisible la preuve de *principe* que l'on cherche à établir, visant uniquement à montrer que le recours à la notion de simulation, pour la validation, n'est pas sans cohérence avec la validation traditionnelle. Nous simplifions donc, dans ce contexte particulier, la validation et la restreignons à une relation de simulation faible.

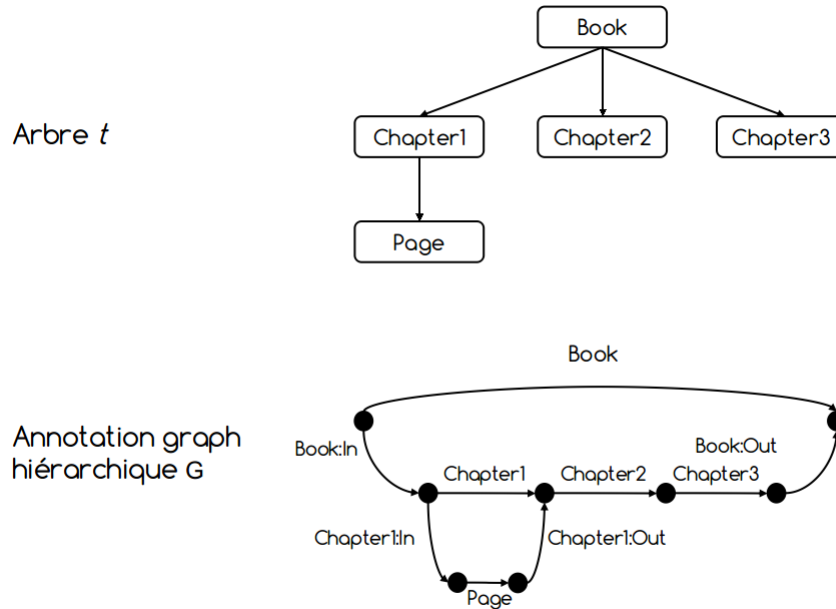
Nous rappelons ici comment représenter un arbre au moyen d'un AGm, suivant la méthode de "hiérarchie structurelle" indiquée par [21], c'est-à-dire au moyen de la notion de domination (voir p. 67).

Nota. Nous préférons ici la notion de domination définie sur les arcs à celle, plus concise mais moins lisible, de domination hiérarchique, où l'arc dominant est effacé, et qui est définie comme une relation sur les nœuds (voir p. 67). La conservation des arcs dominants dans les schémas reflète aussi mieux la forme des règles d'un Tree Automaton, comme nous le verrons.

Definition 12.3 : Dérivation d'un AG depuis un arbre. La dérivation d'un AG depuis un arbre s'appuie sur la correspondance suivante (la Definition synthétique du modèle XML ci-dessous provient de [112] p. 663) :

Arbre t	AGm hiérarchique G
Nœuds labellisés.	Arcs labellisés.
Set ordonné d'enfants (nœuds).	Set ordonné d'arcs dominés par un arc, au sens des AGm (faisant intervenir un couple d'arcs suffixés : <i>In</i> et : <i>Out</i> respectivement).
Pas d'arité préétablie pour les nœuds.	Pas de nombre prédéfini d'arcs dominés pour un arc.
Le texte est représenté par les feuilles.	Le texte est désigné par les références portées par les nœuds.
Racine.	Arc principal liant la racine et la feuille de G .

Exemple. Nous représentons ci-dessous un arbre et son AG équivalent. Nous nous passons de la représentation des références.



Notation. On notera la correspondance entre t et $G : t \stackrel{hier.}{\equiv} G$ (et vice-versa).

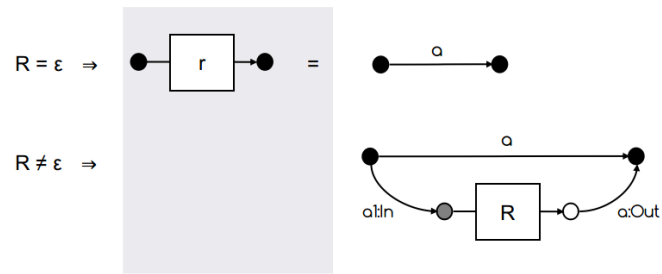
Hypothèse de travail supplémentaire. Nous nous plaçons dans le cas restrictif des schémas où les Définitions de types sont *non récursives*, pour des raisons de simplicité dans la rédaction des preuves. On pourrait étendre le résultat au cas général. Dans ce contexte, la Définition des types peut se faire linéairement selon une approche soit bottom-up (des types inclus vers les types qui les incluent) ou top-down (des types les plus englobants vers les types inclus à l'intérieur). Nous exploiterons implicitement cette propriété dans le développement qui suit.

Etape 1 : Définition d'une construction de schéma S à partir de TA .

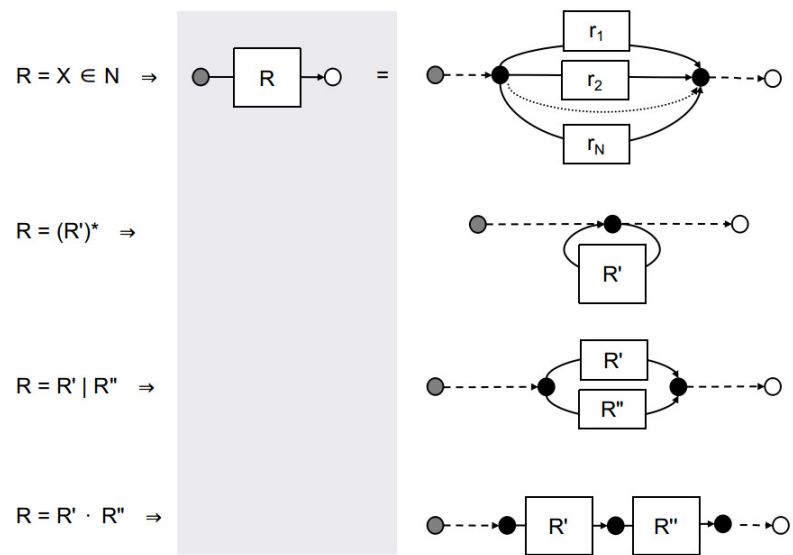
Construisons de manière inductive S à partir de $TA = (N, T, St, P)$.

$\forall s \in S, \exists r \in P \mid r = s \longrightarrow aR.$

On considère une telle règle et on note commodément \boxed{r} le graphe représentant cette règle. Ce graphe est défini ci-dessous, sur la colonne de droite, en fonction de la nature de R :

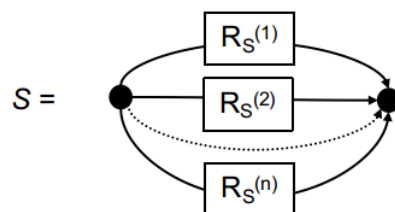


où si $R \neq \epsilon$:



avec $\boxed{r_1} \dots \boxed{r_N}$ les graphes représentant les éléments de P qui sont de la forme $r_i = X \rightarrow \alpha_i R_i$, dans le cas où $R = X$.

Le schéma S complet s'obtient alors de la manière suivante :



où les blocs $\boxed{R_S^{(i)}}$ sont définis comme suit :

- $n = |S|$; $\forall i \in [1; n]$, s_i désigne le i -ième élément de S selon une indexation donnée quelconque ;
- $\forall i \in [1; n]$, $\boxed{R_S^{(i)}}$ représente une expression régulière $R_i = s_i$, de la forme $R_i = X \in N$.
- par conséquent, chaque bloc contient, en parallèle, l'ensemble des blocs $\boxed{r_{ij}}$ définis inductivement plus haut, où $r_i^j = s_i \longrightarrow a_{ij}R_{ij}$.

Notation. On note un couple schéma S / Tree Automata TA obtenu de cette manière : $S \longleftrightarrow TA$.

Definition 12.4 : langage primaire. Soit A un graphe enraciné, possédant une unique feuille et connecté.

On appelle le langage primaire de A le langage défini sur le plus grand sous-graphe de A , possédant la même racine et la même feuille, tel qu'aucun des labels de ses arcs ne possède les suffixes : In ou : Out :

$$lang_{prim}(A) = lang(\max_{|E|} \{G = (V, E) \subset_{leaf(A)}^{root(A)} A; \forall e \in E, \forall l \in \mathcal{L}, label(e) \neq l : In \vee l : Out\})$$

Propriété 12.1 Soit un Tree Automaton $TA = (N, T, St, P)$, soit $r \in P$. Si l'on désigne par $lang_{prim}(\boxed{r})$ le langage primaire du graphe représentant la règle correspondante, comme défini dans la construction inductive de S ci-dessus, alors :

$$\forall r \in P \mid r = X \longrightarrow aR, lang_{prim}(\boxed{r}) = \{a\}.$$

Preuve. Simple vérification sur le graphe représentant r .

Definition 12.5 : fonction $\overset{\blacktriangleright}{T}$ Soit $TA = (T, N, St, P)$.

On définit la fonction $\overset{\blacktriangleright}{T} : N \longrightarrow RE_T$ telle que :

$$\forall X \in N, \overset{\blacktriangleright}{T}(X) \longmapsto \bigcup_{r \in P \mid r = X \longrightarrow aR'} a.$$

Definition 12.6 : extension à une expression régulière Soit R une expression régulière. $\overset{\blacktriangleright}{T}(R)$ est défini de manière inductive traditionnelle de RE_N dans RE_T :

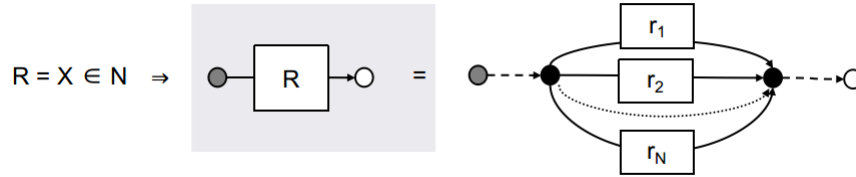
- $R = \epsilon \Rightarrow \overset{\blacktriangleright}{T}(R) = \emptyset$;
- $R = X \in N \Rightarrow \overset{\blacktriangleright}{T}(R) = \overset{\blacktriangleright}{T}(X)$

- $R = (R')^* \Rightarrow \overset{\blacktriangleright}{T}(R) = \overset{\blacktriangleright}{T}(R')^*$
- $R = R'|R'' \Rightarrow \overset{\blacktriangleright}{T}(R) = \overset{\blacktriangleright}{T}(R')|\overset{\blacktriangleright}{T}(R'')$
- $R = R' \cdot R'' \Rightarrow \overset{\blacktriangleright}{T}(R) = \overset{\blacktriangleright}{T}(R') \cdot \overset{\blacktriangleright}{T}(R'')$

Propriété 12.2 Soit $TA = (N, T, St, P)$ définissant une grammaire d'arbre sans type récursif. Quelle que soit R expression régulière apparaissant dans le membre droit d'une règle $r \in P$, si l'on désigne par $lang_{prim}(\boxed{R})$ le langage primaire du graphe représentant l'expression régulière R tel que défini dans la construction inductive de S ci-dessus, alors :

$$lang_{prim}(\boxed{R}) = lang(\overset{\blacktriangleright}{T}(R))$$

Preuve. Par Définition, si $R = X \in N$, la représentation \boxed{R} de R est la suivante :



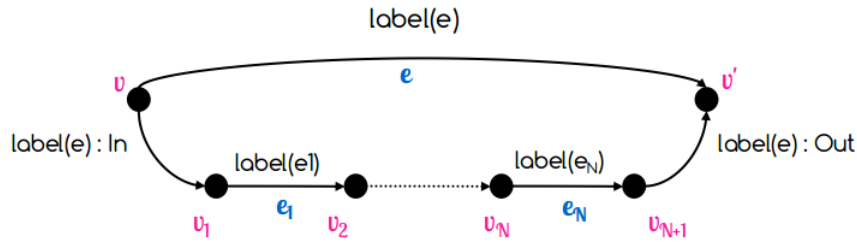
avec $\boxed{r_1} \dots \boxed{r_N}$ les graphes représentant les éléments de P qui sont de la forme $r_i = X \rightarrow a_i R_i$. On en déduit que :

$$\begin{aligned}
 lang_{prim}(\boxed{R}) &= lang_{prim}\left(\bigcup_{r_i = X \rightarrow a R'} \boxed{r_i}\right) \\
 &= \bigcup_{r_i = X \rightarrow a R'} lang_{prim}(\boxed{r_i}), \text{ les sous-graphes } \boxed{r_i} \text{ étant disjoints.} \\
 &= \bigcup_{r_i = X \rightarrow a R'} \{a\} \\
 &= lang(\overset{\blacktriangleright}{T}(X)) \\
 &= lang(\overset{\blacktriangleright}{T}(R))
 \end{aligned}$$

Par induction, on montre que la propriété tient pour $R = (R')^*$, etc. le cas $R = \epsilon$ est trivial.

Etape 2 : reDefinition de la notion d'interprétation, sur la représentation AG des arbres. Soit un Tree Automaton $TA = (N, T, St, P)$ validant un arbre $t = (V, E)$. On adapte la Définition de l'interprétation pour qualifier la relation entre $G = (V_G, E_G) \stackrel{hier.}{\equiv} t$ et $TA : \exists I : E_G \rightarrow N$ telle que:

1. pour $e_0 \mid \text{root}(G) \lfloor e_0 \rfloor \text{leaf}(G)$, on a $I(e_0) \in S$;
2. quel que soit le graphe de la forme



inclus dans G , $\exists r \in P$;

- $r = X \longrightarrow aR$;
- $I(e) = X$;
- $\text{label}(e) = a$;
- $I(e_1) \cdot \dots \cdot I(e_N) \in \text{lang}(R)$.

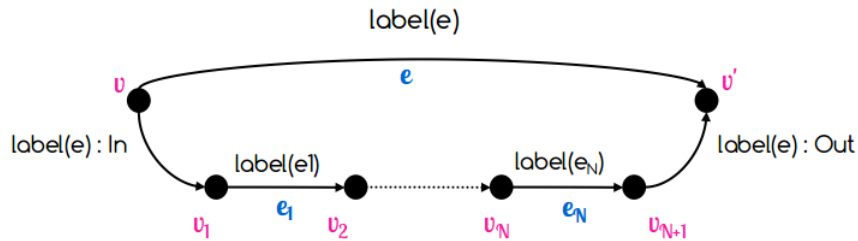
Etape 3 : Etablissement de la relation entre Interprétation et Simulation.

On veut montrer que :

- Etant donné $TA = (N, T, St, P)$ un Tree Automaton, $t = (V, E)$ un arbre,
- Si l'on se donne $G = (V_G, E_G)$ un Annotation graph tel que $G \stackrel{\text{hier.}}{\equiv} t$,
- Si l'on se donne S un schéma tel que $S \longleftrightarrow TA$,
- Alors $S \leftrightarrow G$.

Preuve. PARTIE 1 :

On a vu que $\exists I(G)/TA$ implique que quel que soit le graphe de la forme :



inclus dans G , $\exists r = X \rightarrow aR \in P$ telle que $\prod_{i \in [1;N]} I(e_i) \in \text{lang}(\boxed{R})$.

$$\text{Or } \forall i \in [1;N], \text{label}(e_i) \in \text{lang}(\overset{\blacktriangleright}{T}(I(e_i))) = \bigcup_{r \in P \mid r = I(e_i) \rightarrow aR'} a.$$

En outre, $\overset{\blacktriangleright}{T}(I(e_1) \cdot \dots \cdot I(e_N)) = \overset{\blacktriangleright}{T}(I(e_1)) \cdot \dots \cdot \overset{\blacktriangleright}{T}(I(e_N))$ par Definition de $\overset{\blacktriangleright}{T}$ sur une expression régulière de non terminaux.

$$\text{On en déduit que } \prod_{i \in [1;N]} \text{label}(e_i) \in \text{lang}(\overset{\blacktriangleright}{T}(\prod_{i \in [1;N]} I(e_i))).$$

$$\text{Par ailleurs, } \prod_{i \in [1;N]} I(e_i) \in \text{lang}(R) \Rightarrow \{ \prod_{i \in [1;N]} I(e_i) \} \subseteq \text{lang}(R).$$

$$\Rightarrow \{ \overset{\blacktriangleright}{T}(I(e_1)) \cdot \dots \cdot \overset{\blacktriangleright}{T}(I(e_N)) \} \subseteq \text{lang}(\overset{\blacktriangleright}{T}(R))$$

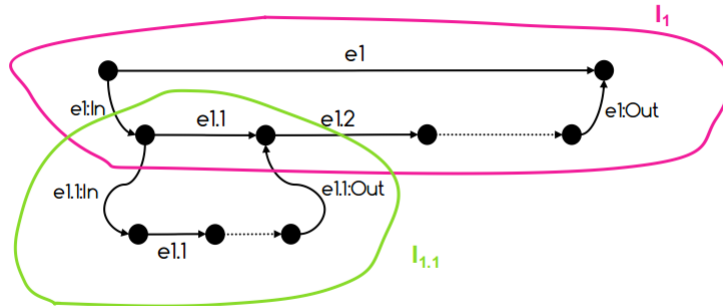
$$\Rightarrow \text{lang}(\overset{\blacktriangleright}{T}(\prod_{i \in [1;N]} I(e_i))) \subseteq \text{lang}(\overset{\blacktriangleright}{T}(R)) = \text{lang}_{\text{prim}}(\boxed{R})$$

D'où $\prod_{i \in [1;N]} \text{label}(e_i) \in \text{lang}(\overset{\blacktriangleright}{T}(\prod_{i \in [1;N]} I(e_i))) \subset \text{lang}(\overset{\blacktriangleright}{T}(R)) = \text{lang}_{\text{prim}}(\boxed{R}) \subseteq \text{lang}(\boxed{R})$,

$$\text{Soit finalement } \prod_{i \in [1;N]} \text{label}(e_i) \in \text{lang}(\boxed{R})$$

PARTIE 2.1 :

On cherche maintenant à montrer que $I(t)/G \Rightarrow S \hookrightarrow G$. Pour cela, nous allons nous appuyer sur la structure suivante représentative de G :



On va définir par récursion un sous-graphe de S , noté S_* , contenant uniquement les chemins de S qui sont instanciés dans G ; on va également définir récursivement la hiérarchie G_* qui correspond à la forme linéarisée de S_* , dont le langage contiendra le langage de G . Nous exploiterons ces deux graphes intermédiaires pour conclure quand à la simulation de G par S .

On sait que $\exists I(G)/TA$. Aussi, $\exists r_S \in P$ telle que $r_S = I(e_1) \rightarrow \text{label}(e_1)R_1$.

α) Cas où $R_1 = \epsilon$

$$R_1 = \epsilon \Rightarrow t = \text{root}(G)[e_1]\text{leaf}(G) \text{ et } \text{lang}(G) = \{\text{label}(e_1)\}.$$

$$\text{En outre } r_S \in P \wedge s \in St \Rightarrow \text{root}(S)[\epsilon][\boxed{r_S}][\epsilon]\text{leaf}(S) \subset_{\text{leaf}(S)}^{\text{root}(S)} S$$

Or $label(e_1) \in lang_{prim}(\boxed{r_S}) \Rightarrow label(e_1) \in lang_{prim}(S)$.

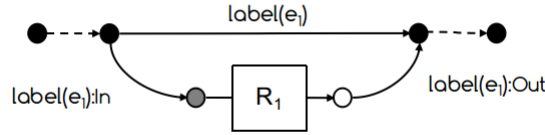
D'où $root(S) \mid \epsilon \mid G \mid \epsilon \mid leaf(S) \subset_{leaf(S)}^{root(S)} S$

Et donc $S \hookrightarrow G$.

β) Cas où $R_1 \neq \epsilon$

De la même manière, on a $label(e_1) \in lang_{prim}(S)$.

En outre, $r_S = s \rightarrow label(e_1)R_1 \mid s \in St \Rightarrow S_1 \subset_{leaf(S)}^{root(S)} S$, si l'on note S_1 le graphe ci-dessous :



S_1 ne contenant pas de cycle, sa forme linéarisée lui est égale. On la note G_1 .

Or d'après la PARTIE 1 ci-dessus, $\prod_{i \in [1;N]} label(e_{1,i}) \in lang_{prim}(\boxed{R_1})$

$$\begin{aligned} \Rightarrow lang(I_1) &= \{label(e_1) : In \cdot \prod_{i \in [1;N]} label(e_{1,i}) \cdot label(e_1) : Out\} \\ &\subseteq \{label(e_1)\} \cup \{label(e_1) : In\} \cdot lang_{prim}(\boxed{R_1}) \cdot \{label(e_1) : Out\} \\ &\subseteq lang(G_1) \end{aligned}$$

Soit finalement : $lang(I_1) \subseteq lang(G_1)$.

PARTIE 2.2 :

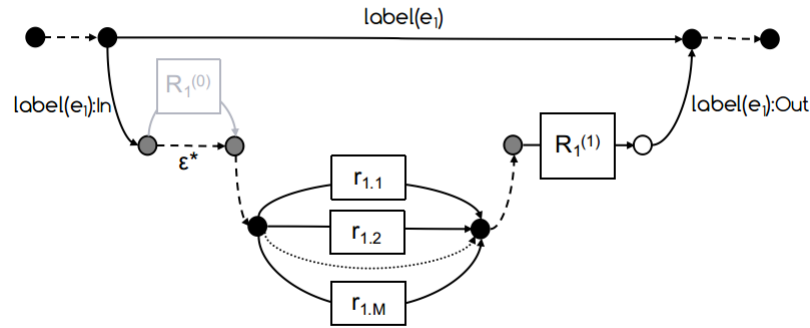
On va alors initier la récurrence en montrant que $lang(I_1 \bigcup_{i \in [1;N]} I_{1,i}) \subseteq lang(G_*)$.

On considère tout d'abord uniquement $I_{1,1}$

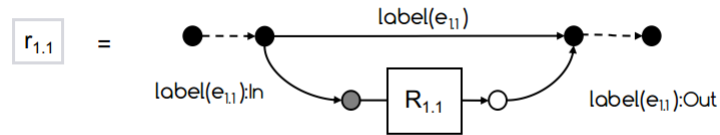
$R \neq \epsilon \Rightarrow \exists r_{1,1} = X_{1,1} \rightarrow a_{1,1}R_{1,1}$ telle que :

1. on peut écrire R_1 sous la forme $R_1 = R_1^{(0)} \mid (R_1^{(0)} \cdot X_{1,1} \cdot R_1^{(1)})$, $R_1^{(1)} \in RE_N \wedge X_{1,1} \neq \epsilon$;
2. $I(e_{1,1}) = X_{1,1}$, $label(e_{1,1}) = a_{1,1}$ et $\prod_i I(e_{1,1,i}) \in lang(R_{1,1})$ (du fait de l'existence d'une interprétation de G).

Du point 1 ci-dessus, on déduit que le graphe S_1^+ représenté ci-dessous vérifie $S_1^+ \subset_{leaf(S)}^{root(S)} S_1$:



avec $\boxed{r_{1,1}}$ réduit à un arc labellisé $label(e_{1,1})$ si $R_{1,1} = \epsilon$ ou sinon :



D'après la PARTIE 2.1 de cette preuve, on déduit que :

$$lang(I_{1,1}) \subseteq \{label(e_{1,1})\} \cup \{label(e_{1,1}) : In\} \cdot lang_{prim}(\boxed{R_{1,1}}) \cdot \{label(e_{1,1}) : Out\}.$$

On en déduit que $lang(G) \subseteq lang(G_{1,1}^{(1,1)})$, avec $G_{1,1}^{(1,1)}$ représenté dans la table 12.1 page 338.

Nota. L'illustration que l'on propose de $S_{1,1}$ est déjà une linéarisation du sous-graphe $S_{1,1} \subseteq S_1$: un même nœud peut y apparaître deux fois (cela pourrait se produire, par exemple, dans certains cas où une boucle de R_1 se traduirait par un pattern répété un nombre fini de fois dans G). Idem pour les autres sous-graphes de S . Dans le cas où une séquence d'arcs consécutifs de G correspondraient à l'instanciation, N fois successivement, d'un cycle de S , alors $S_{1,L}$ comporterait N fois, linéairement, la même séquence. $G_{1,L}$ est le graphe obtenu en ôtant les arcs epsilon, et en attribuant un identifiant différent à chaque occurrence d'un même nœud, dans le cas de nœuds répétés dans $S_{1,L}$.

On applique alors ce raisonnement par récurrence horizontale, pour obtenir une qualification en termes de langage de $\bigcup_{i \in [1;N]} I_{1,i}$, en faisant varier i , comme cela est

illustré dans la table 12.1 page 338 :

α) Si $R_1^{(1)} = \epsilon$, on passe directement à la récurrence en profondeur (voir la PARTIE 2.3 ci-dessous).

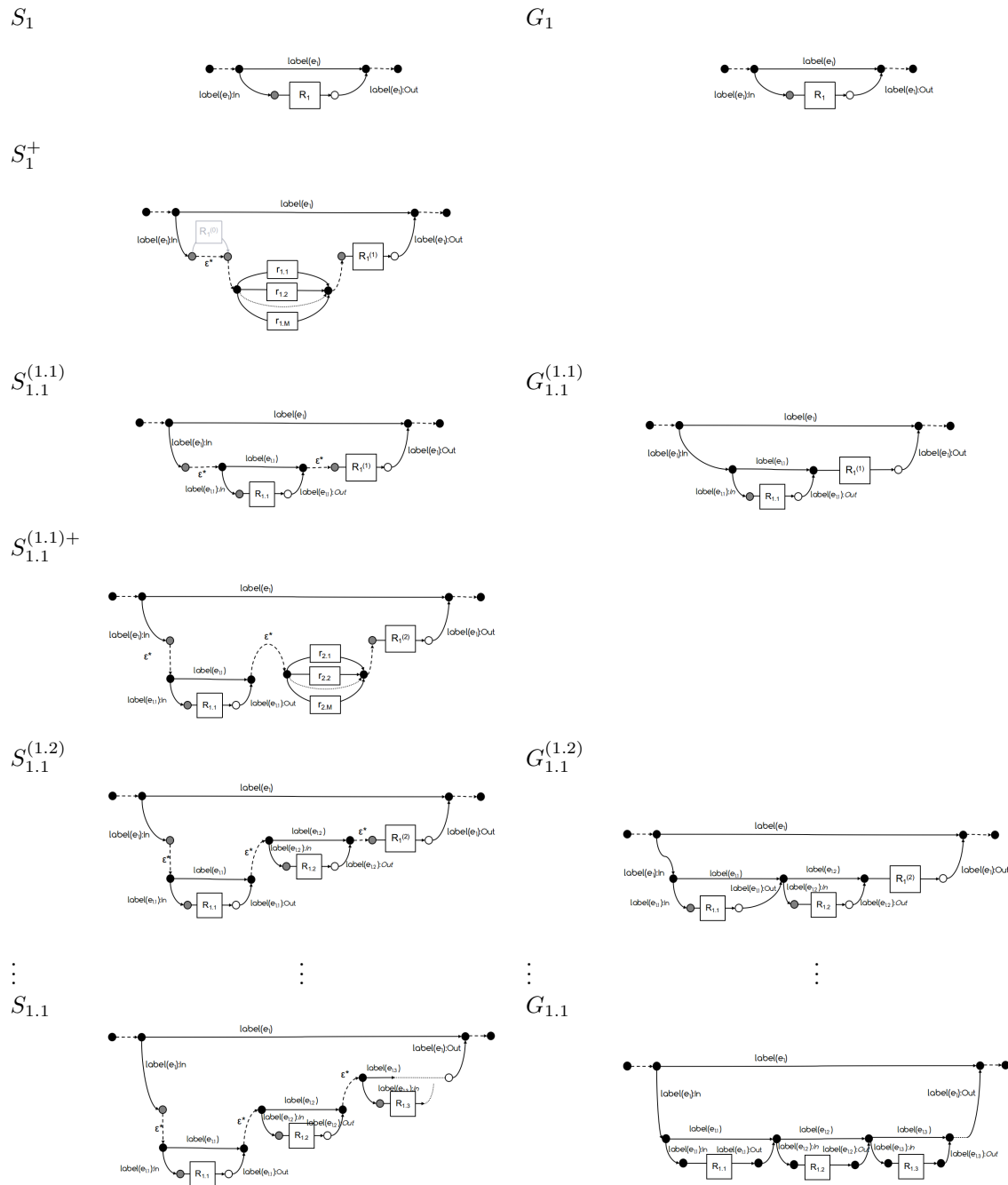
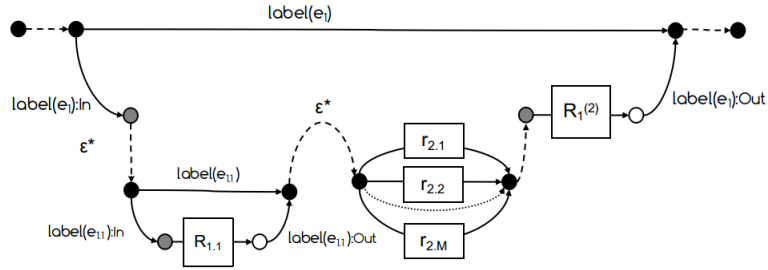


Table 12.1: Bilan des sous-graphes de S définis par récurrence horizontale, par ordre décroissant sur l'inclusion ($S_1 \supseteq S_1^+$ etc.), et des hiérarchies correspondantes.

β) Si $R_1^{(1)} \neq \epsilon$, alors R_1 peut se réécrire $R_1 = R'_1 | (R_1^{(0)}? \cdot X_{1.1}(?) \cdot R_1^{(2)}? \cdot X_{1.2}(?) \cdot R_1^{(3)})$,
 $R_1^{(2)} \in RE_N \wedge X_{1.2} \neq \epsilon$.

Cela revient à considérer le graphe $S_{1.1}^{(1.1)+}$ ci-dessous, qui vérifie $S_{1.1}^{(1.1)+} \subset_{leaf(S)}^{root(S)} S_{1.1}^{(1.1)}$.



Avec une notation cohérente avec ce qui précède, on montre de même que :

$$lang(I_{1.2}) \subseteq \{label(e_{1.2})\} \cup \{label(e_{1.2}) : In\} \cdot lang_{prim}(\boxed{R_{1.2}}) \cdot \{label(e_{1.2}) : Out\}.$$

Ce résultat est vrai pour les valeurs de i suivantes indifféremment, jusqu'à épuisement de R_1 .

On en déduit que :

$$lang\left(\bigcup_{i \in [1;N]} I_{1.i}\right) \subseteq \prod_{i \in [1;N]} \left\{label(e_{1.i})\right\} \cup \{label(e_{1.i}) : In\} \cdot lang_{prim}(\boxed{R_{1.i}}) \cdot \{label(e_{1.i}) : Out\}$$

D'où $\{label(e_1) : In\} \cdot \prod_{i \in [1;N]} \left\{label(e_{1.i})\right\} \cup \{label(e_{1.i}) : In\} \cdot lang_{prim}(\boxed{R_{1.i}}) \cdot \{label(e_{1.i}) : Out\} \cdot \{label(e_1) : Out\} \subseteq lang(G_{1.1})$, avec $G_{1.1}$ tel que représenté dans la table 12.1 page 338.

$$\text{Soit } lang(I_1 \bigcup_{i \in [1;N]} I_{1.i}) \subseteq lang(S_{1.1}).$$

PARTIE 2.2 Récurrence en profondeur.

Pour finir la preuve, il faut en plus, pour chaque $i \in [1;N]$, considérer $[I_{1.i.j}]_j$:

Cela revient à répéter exactement toute ces PARTIE 2.2-3 non plus à $I_{1.1}$, mais à $I_{1.i.j}$. On achève la récurrence en ayant construit S_* et G_* de manière analogue à $S_{1.1}$ et $G_{1.1}$, tels que $S_* \subset_{leaf(S)}^{root(S)} S$ et $lang(G) \subseteq lang(G_*)$.

CONCLUSION INTERMEDIAIRE. Soit K la profondeur de $t \stackrel{hier.}{\equiv} G$.

Pour conclure, il faut remarquer que :

$$I_1 \bigcup_{i(1) \in [1;N(1)]} (I_{1.i(1)} \bigcup_{i(2) \in [1;N(2)]} (I_{1.i(1).i(2)} \dots \bigcup_{i(K) \in [1;N(K)]} I_{1.i(1)\dots i(K)})) = G.$$

L'itération de la PARTIE 2 de cette preuve conduit donc au résultat suivant :

$$lang(I_1 \bigcup_{i(1) \in [1;N(1)]} I_{1.i(1)} \bigcup_{i(2) \in [1;N(2)]} I_{1.i(1).i(2)} \dots \bigcup_{i(K) \in [1;N(K)]} I_{1.\dots i(K)}) \subseteq lang(G_*),$$

c'est-à-dire $lang(G) \subseteq lang(G_*)$. Et on peut vérifier que G_* est une hiérarchie.

PARTIE 3

Pour conclure, nous remarquons que l'on a établi le résultat suivant : Il existe une hiérarchie $G_* \subseteq S$ tel que $lang(G) \subseteq lang(G_*)$, et il existe $S^* \subseteq_{leaf(S)}^{root(S)} S$, tels que S_* correspond, en termes informels, à une G_* dont on aurait "fusionné certains nœuds" (nous préciserons cette idée ultérieurement).

Nous concluons au moyen des trois propriétés suivantes. Les deux premières établissent que si deux hiérarchies partagent une partie de leur langage, alors la plus petite est incluse dans la plus grande. On en déduit que dans le cas de hiérarchies, $lang(A) \subseteq lang(B) \Leftrightarrow B \hookrightarrow A$.

On montre ensuite, en définissant les notions formellement, que si un graphe C est obtenu en "fusionnant certains nœuds" à partir d'un autre graphe B , alors $C \hookrightarrow B$.

Par transitivité, nous obtiendrons $C \hookrightarrow A$, soit pour nous $S_* \hookrightarrow G$, puis nous concluons en exploitant l'inclusion de S_* dans S , ce qui donne le résultat final : $S \hookrightarrow G$.

Propriété 12.3 Soient A, B des AGm hiérarchiques tels que définis précédemment, tels que $lang(B) \subseteq lang(A)$. Alors $A \subseteq_{leaf(B)}^{root(B)} B$.

Preuve. Cela tient à la manière unique dont est définie une hiérarchie, pour un arbre donné. Or seuls deux arbres différents peuvent donner deux vocabulaires différents.

Corollaire. Soient A, B des AGm hiérarchiques. Alors :

$$lang(B) \subseteq lang(A) \Rightarrow A \hookrightarrow B.$$

Preuve. Conséquence immédiate de l'inclusion (avec partage des racines).

Definition et propriété 12.7 : Graphe obtenu en fusionnant deux nœuds d'un autre graphe. Soient $A = (V_A, E_A), B = (V_B, E_B)$ tels que :

- $V_A = \{v_x^A, v_y^A; v_i^A\}_{i \in I \subset \mathbb{N}}, E_A = \{e_j^A\}_{j \in J \subset \mathbb{N}}$;
- $V_B = \{v_x^B, v_i^B\}_{i \in I \subset \mathbb{N}}, E_B = \{e_j^B\}_{j \in J \subset \mathbb{N}}$;
- et $\forall j \in J, label(e_j^A) = label(e_j^B)$.

et vérifiant les conditions suivantes, $\forall j \in J$:

1. $\exists i_s, i_e \in I \mid (sut(e_j^A) = v_{i_s}^A; end(e_j^A) = v_{i_e}^A) \in (V_A \setminus \{v_y^A\})^2$
 $\Leftrightarrow (sut(e_j^B, end(e_j^B)) = (v_{i_s}^B; v_{i_e}^B)$
2. $sut(e_j^A) = v_y^A \Leftrightarrow sut(e_j^B) = v_x^B$
3. $end(e_j^A) = v_y^A \Leftrightarrow end(e_j^B) = v_x^B$

Alors on dit que B est le graphe obtenu en fusionnant v_x^A et v_y^A dans A .

Ces deux graphes vérifient la propriété suivante : $B \hookrightarrow A$. *Preuve.* On vérifie que $D = \{(v_i^A, v_i^B), (v_x^A, v_x^B), (v_y^A, v_x^B)\}_{i \in I}$ définit une simulation de A par B .

Nota. Cette propriété est généralisable aux graphes obtenus en fusionnant plusieurs nœuds d'un graphe, par transitivité de la simulation.

FIN DE LA PREUVE.

Chapter 13

eAG *Mod* Operation: General Case

Nous décrivons ici les principaux algorithmes qui seront mis en œuvre à la fois pour la dérivation d'une modification dans le domaine des instances à partir d'une modification définie dans le domaine des schémas, et pour bidirectionnaliser la première. À ce titre, nous rappelons ci-dessous le processus par lequel la dérivation s'opère, et les calculs qui peuvent ensuite être réalisés lorsque deux graphes liés par une modification bidirectionnelle sont mis à jour :

La situation est la suivante : Soit une étampe G_i , deux cellules schématiques χ et χ^+ permettant de définir une modification $Mod(\chi, \chi^+)$ telle que $Mod(\chi, \chi^+).G_i = G_{i+1} \neq G_i$.

Soit un graphe H_i sans boucle ni partie connexe réduite à un point, et dont les nœuds ne possèdent chacun qu'une valeur de type, tel que $[Temp.G_i]$ soit exhaustif sur H_i . H_{i+1} , image de H_i par la modification dérivée de $Mod(\chi, \chi^+)$, va être obtenu de la manière suivante :

1. La matrice-bloc $[Temp.\chi]$ est analysée pour déterminer son pavage, c'est-à-dire pour identifier les ensembles de matrices négatives et les ensembles de matrices positives qui sont susceptibles de contenir des valeurs désignant les mêmes nœuds, et devant donc faire l'objet d'un alignement local, au moyen d'une permutation σ_k , dans la perspective de rechercher les couples racine-feuille liés par un chemin instanciant χ dans H_i .
2. Ensuite, sur le pavage obtenu, dans lequel les variables matricielles sont remplacées par leur valeur dans H_i , les permutations locales sont déterminées.
3. Cet ensemble de permutations est lu de sorte à fournir la liste des paires racine-feuilles de sous-graphes linéaires de H_i qui instaicient un chemin de χ .
4. Une instance d'une cellule schématique (ici : χ^+) doit alors être créée entre les nœuds formant une telle paire.

A ce moment, on dispose d'un nouveau graphe H_{i+1} , image de H_i . L'un et l'autre graphe peuvent alors faire l'objet de mises à jour nécessitant de mettre en œuvre le caractère bidirectionnel de la dérivée de $Mod(\chi, \chi^+)$: il s'agit dans ce cas soit d'insertion de nouvelles instances d'une cellule schématique dans l'instance correspondante, soit de leur suppression.

1. Dans le cas d'une insertion : seules les insertions venant augmenter la liste des couples racines-feuilles d'instances de χ . Dans ce cas, le processus est similaire au point 3 ci-dessus, à cela près que les matrices ne sont pas créées, mais ajoutées, dans leur forme infinie, à celles qui existent dans l'autre instance. Nous ne développons pas davantage.
2. Dans le cas d'une suppression, un algorithme de propagation particulier doit être mis en œuvre. Celui-ci reste à définir.

13.1 Pavage

Nous avons illustré la détermination du pavage sur le cas le plus simple : celui où χ n'est constitué que d'un unique chemin non cyclique. Le cas général se rapporte aux cellules comportant plusieurs chemins cycliques.

La multiplicité des chemins est facile à prendre en compte étant donné que, par définition des cellules, deux chemins ne possédant pas le même premier arc sont d'intersection nulle : un traitement en parallèle des plus grands sous-graphes de χ partant de la racine et conduisant à la feuille et identifiés par un unique arc issu de la racine (sous-graphes qui sont eux-même des cellules) est donc possible.

Reste la question des cycles. Celle-ci complexifie en revanche la procédure de détermination du pavage, par rapport à l'exemple illustré plus haut. Effectivement, dans cet exemple, les matrices du pavage s'enchaînaient une à une, depuis l'unique matrice décrivant les racines possibles des sous-graphes de I_S instanciant un chemin de χ , si bien que, par la suite, l'alignement opéré par les permutations σ_k s'opérait entre les colonnes d'une matrice négative et celles d'une matrice positive, constituant à elles deux l'ensemble des matrices non nulles d'une certaine ligne-bloc de $[Temp.\chi]$. Dans le cas général, du fait que les cycles augmentent la connectivité de certains nœuds (un cycle impliquant au moins une bifurcation et une convergence de chemins), plusieurs matrices positives (ou négatives) peuvent se trouver sur une même ligne-bloc. Aussi, les matrices du pavage, dans le cas général, ne s'enchaîneront pas de manière unitaire. Nous illustrerons cela après avoir défini l'algorithme de pavage que nous utiliserons.

Algorithme. On a les données suivantes :

- $[Temp.\chi]$ le *Template* de la cellule de sélection χ ;
- i_r l'indice de la seule ligne-bloc de $[Temp.\chi]$ ne comportant pas de variable matricielle négative.
- $\{J_r\}$ une liste recouvrant l'ensemble des indices des colonnes-bloc non nulles de la ligne-bloc d'indice i_r de $[Temp.\chi]$. Les indices de cette liste débutent conventionnellement à 0.

On se donne les structures de données suivantes :

- Le pavage P à remplir, qui sera la sortie de l'algorithme. Il s'agit d'une liste de piles.
Pour alimenter le pavage, on définit $insert.P([M], h)$, qui insère $[M]$ à la fin de la pile contenue à la position h de la liste P . Les indices de cette liste débutent conventionnellement à 0.
- Un dictionnaire F qui associe des drapeaux à des valeurs d'indice (l'algorithme procédant à l'affectation de drapeaux à des colonnes-bloc de $[Temp.\chi]$.
L'accès à la valeur du drapeau associé à un indice x se fait au moyen d'une fonction¹ $flag : flag(x)$ retourne soit cette valeur, si elle existe, soit une valeur nulle.
L'insertion d'un couple indice-drapeau dans le dictionnaire se fera au moyen d'une opération $insFlag$, prenant pour unique paramètre la valeur de l'indice : le drapeau, lui, sera créé de sorte à être unique dans le dictionnaire².
- Une pile notée Mmo , dont les éléments sont des couples $([M], h)$, où $[M]$ désigne une matrice et h un indice de la liste P .
On insère en fin de pile un couple à l'aide de l'opération $push.Mmo$; on récupère et supprime le dernier élément de la pile à l'aide de l'opération $pop.Mmo$.

On se donne les opérations supplémentaires suivantes :

- $col([M])$ dont l'argument est une variable matricielle $[M]$ de $[Temp.\chi]$, et qui retourne la valeur de l'indice de la colonne-bloc de cette variable dans $[Temp.\chi]$.
- une opération permettant d'affecter en exposant une valeur de drapeau à une variable matricielle. cette opération n'est pas représentée par un symbole : on notera $[M]^f$ le résultat de son application à la variable matricielle $[M]$ avec le drapeau f .

On définit les variables suivantes :

- h^* la hauteur courante dans le pavage P ;
- l^* la ligne courante dans $[Temp.\chi]$, initialisée à la valeur i_r ;
- j^* la colonne courante dans $[Temp.\chi]$;
- $[A^*]$ une matrice positive ($[A^*] > [0]$) ;
- $[B^*]$ une matrice négative ($[B^*] < [0]$).

¹Effectivement, une seule valeur de drapeau au plus pourra être affectée à une valeur d'indice donnée.

²Autrement dit, toute nouvelle entrée de couple dans le dictionnaire s'accompagne de la création d'une nouvelle valeur de drapeau.

On initialise l'algorithme : $push.Mmo([Temp.\chi]_{i_r, \{J_r\}_0}, 0)$.

Algorithm 5: CONSTRUCTION DU PAVAGE P .

```

begin
  while  $Mmo \neq \emptyset$  do
    for  $j^*$  ranging from 0 to  $|\{J_r\}| - 1$  do
       $([A^*], h^*) = pop.Mmo$  ;
       $insert_P([A^*]^{flag(col([A^*])}), h^*)$  ;
      if  $flag(col([A^*])) = \emptyset$  then
         $h^* = h^* + 1$  ;
        Find  $[B^*] = [[Temp.\chi]_{i', col([A^*])}]$  the unique matrix variable on the
        block-column  $col([A^*])$  ;
         $i^* = i'$  ;
         $insert_P([B^*], h^*)$  ;
        forall the
           $[B'] = [[Temp.\chi]_{i^*, j' \neq col([B^*])}] \mid [[Temp.\chi]_{i^*, j' \neq col([B^*])}] < [0]$  do
             $insFlag(col([B']))$  ;
             $insert_P([B']^{flag(col([B']))}, h^*)$  ;
           $h^* = h^* + 1$  ;
          forall the  $[A'] = [[Temp.\chi]_{i^*, j}] \mid [[Temp.\chi]_{i^*, j}] > [0]$  do
             $push.Mmo([A'], h^*)$  ;

```

13.2 Permutations entre matrices d'une même instance

On a à notre disposition :

- $\{I_S/[Temp.\chi]\}$ la liste des valeurs (extraites de $\{I_S/[Temp.S]\}$) affectées aux variables matricielles de $[Temp.\chi]$. A partir de cette donnée, on peut disposer d'un pavage noté P_I , construit à partir de P dans lequel les valeurs matricielles remplacent les variables correspondantes.
- La taille de la liste P établie précédemment : $length.P$. La taille de P_I est identique. On en déduit l'indice maximal : $H = length.P - 1$.
Pour toute case $(P_I)_h$, on connaît aussi l'indice du dernier élément de la pile contenue dans la case : $j(h)$.
- Le dictionnaire F établi précédemment.

On va maintenant analyser P_I et définir les permutations qui articulent chaque ligne de P_I comportant des matrices négatives (à part la ligne décrivant les feuilles) et la ligne suivante, positive. Effectivement, ces paires de lignes, quand elles existent, répètent les mêmes nœuds, sans exception, si un arc parvient à un nœud v_m apparaissant sur la ligne négative, et qu'il existe une ligne positive en regard, alors un arc doit être issu de ce nœud car ce nœud n'est pas une feuille, ce qui implique que v_m apparaisse dans au moins une matrice de la ligne positive. Ce que l'on cherche, c'est à déterminer la permutation, dans le sens que l'on a défini dans l'exemple plus haut, qui aligne les différentes colonnes des différentes matrices des lignes positive et négative où apparaît ce nœud v_m .

Pour ce faire, nous n'allons pas considérer une colonne comme une colonne dans une matrice de la ligne : nous allons considérer la liste des colonnes des matrices de chaque ligne, et aligner les colonnes de ces deux ensembles.

Nous allons pour cela appliquer deux algorithmes : le premier fournira les paires de listes de nœuds (i.e. de colonnes) à aligner ; le second donnera les permutations réalisant ces alignements locaux.

Algorithm 6: DÉTERMINATION DES PAIRES DE LISTES DE NŒUDS À ALIGNER

```

begin
  forall the  $h \in [0; H]$  do
    Create an empty stack  $Line_h$  ;
    forall the matrix  $[M_j] = [(P_I)_h]_j = [B_{xy}] \vee [A_{xy}]$  defined by  $j \in [0; j(h)]$  do
      Search for the couples of naturals  $(m, n) \in \mathbb{N}^2$  so that  $[M_j]_{n,m} \neq [0]$  ;
       $push.Line_h(n^{flag([M_j])}, x)$  ;
      /* This means we fill  $Line_h$  with the value of the identifier of the
         nodes that appear on the  $h^{th}$  position of  $P_I$ , marked by the flag of
         the matrix they belong to, if such a flag exists, together with the
         value of their type (given by the line-block index  $x$  of the matrix
         they belong to, in  $[I_S/[Temp.\chi]]$ . */

```

Algorithm 7: ÉTABLISSEMENT DES PERMUTATIONS SUR LE PAVAGE.

```

begin
  forall the  $h = 2n + 1, h < H$  do
     $L_B = Line_h$  ;
     $L_A = Line_{h+1}$  ;
     $\sigma_h = ([ ], [ ]) ;$ 
    /* Here we initialize a pair of stacks that will eventually be a
       permutation  $\sigma_h$ . The first stack in it, denoted  $\sigma_{B,h}$  hereafter,
       identified by the index 0, will contain the elements of  $L_B$ , repeated if
       needed ; the second one, denoted  $\sigma_{A,h}$  hereafter, will contain the
       elements of  $L_A$ , repeated if needed, in an order dictated by  $\sigma_{B,h}$ . */
    1 forall the indexes  $j$  of  $L_B$  do
    2        $n = 0$  ;
    3       forall the indexes  $k$  of  $L_A$  do
    4         Let us note  $\overline{([L_A]_k)_0}$  the node identifier  $([L_A]_k)_0$  without considering its
           flag, if it has one ;
    5         Let us note  $\overline{([L_B]_j)_0}$  the node identifier  $([L_B]_j)_0$  without considering its flag,
           if it has one ;
    6         if  $\overline{([L_A]_k)_0} = \overline{([L_B]_j)_0}$  then
    7            $push.\sigma_{B,h}(j^{flag(( [L_B]_j)_0)}(*), ([L_B]_j)_1)$  ;
    8            $push.\sigma_{A,h}(k^{flag(( [L_A]_k)_0)}, ([L_A]_k)_1)$  ;
    9            $n = n + 1$  ;
           /* NB : Here, only the identifiers that appear both in an A and a
              B matrix (at least) on a certain block-line of  $[\chi/[Temp.\chi]]$  are
              taken into account : it means that the nodes that are only
              pointed at by exogeneous edges will not be considered for
              alignment - same for nodes that are summit of exogeneous edges
              only. */
    10        forall the indexes  $k$  of  $\sigma_{A,h}$  do
    11          Determine  $m = Card(\{([\sigma_{A,h}]_l)_0 = ([\sigma_{A,h}]_k)_0 \mid l < k\})$  ;
    12          Replace  $([\sigma_{A,h}]_k)_0$  by  $([\sigma_{A,h}]_k)_0(*)^m$  ;

```

13.3 Alignement

Pour finir, nous devons procéder à la lecture des permutations, de sorte à en tirer les paires racine-feuille.

On se donne :

- Une opération permettant d'associer un tag x à la première valeur d'un couple $N = (n, t)$. Nous représentons cette opération par l'opérateur $\leftarrow-$, de sorte que $N \leftarrow x$ représente à la fois l'affectation et la valeur qui en résulte.
- Une opération qui occulte la partie étoilée d'une valeur extraite de $\sigma_{B,h}$ ou $\sigma_{A,h}$: $\forall n, m, t, \overline{(n(*)^m, t)} = (n, t)$.
- Une pile de piles. On désigne la pile principale par Rs . Les sous-piles sont indexées : on les désignera par $Rs_j, j \in \mathbb{N}$.
On se donne la fonction $getIndexes$, de sorte que $getIndexes.Rs(X)$ retourne une liste d'indices, ceux des sous-piles de Rs dont le dernier élément est un couple, dont le premier élément est X .

On note P le nombre permutations $\sigma_h : h$ varie dans $[1; P]$.

Algorithm 8: ETABLISSEMENT DES PERMUTATIONS SUR LE PAVAGE.

```

begin
  forall the  $h$  from 1 to  $P$  do
    /* Here, we reverse crawl the first line of the  $h^{th}$  permutation, because
       starred values, which indicate that a given node is to be aligned with
       more than one node (i.e. has more than one immediate descendants
       within the instance of the cell), will be discovered first. Since  $R_s$ 
       is made to contain the ordered lists of all the ancestors of a node
       along the paths it pertains to, for starred nodes, the aforementioned
       lists must be copied if we want to stick to linear representation of
       lineages, before being completed by the node's descendants'
       description. Since the starred values appear first, there is no need
       to check, for each value, if the ancestor's lists must be copied or
       not. If the value is starred, it must, else, there is no need.      */
1   forall the indexes  $k$  of  $\overline{\sigma_{B,h}}$ , in decreasing order do
2      $J = \{getIndexes.Rs((\sigma_{B,h})_k \leftarrow N)\}_{N < h}$ ;
     /*  $J = \emptyset$  means that the node corresponding to the current index of the
        permutation has not been encountered yet. If the node is flagged,
        then it means it is part of an instance of a cycle in the cell : it
        must be considered. If it is not flagged and  $h \neq 1$ , then the node
        is the end of an exogeneous edge, and must not be considered.
        Finally, the set of the non-flagged nodes so that  $h = 1$  contains the
        roots of the paths instanciating a path of the cell : the
        so-characterized nodes must be considered ; the real such roots will
        be sorted out later, as the ones leading to a node whose type is the
        one of the leaf of the cell (see the procedure after this algo).
        */
3     if  $J = \emptyset$  and, in case  $h \neq 1$ ,  $flag((\sigma_{B,h})_k)_1 \neq 1$  then
4       Be a stack  $S = [(\sigma_{B,h})_k; (\sigma_{A,h})_k \leftarrow h]$ ;
5       push.Rs( $S$ );
6     else
7       if  $\overline{(\sigma_{B,h})_k}_0 \neq (\sigma_{B,h})_k_0$  (i.e. the index value (vs. the type value) in  $\sigma_{B,h}$ ) $_k$ 
          is starred) then
8         forall the  $j \in J$  do
9           push.Rs( $Rs_j$ );
           /* Here we copy the stacks of  $R_s$  ending by a couple
              containing  $\overline{(\sigma_{B,h})_k}_1 \leftarrow N, N < h$ .      */
10        forall the  $j \in J$  do
11          push.Rs( $Rs_j((\sigma_{A,h})_k \leftarrow h)$ );
           /* Here we insert  $\overline{(\sigma_{A,h})_k} \leftarrow h$  in the lines whose index belongs to
               $J$ , i.e. the lines ending by  $\overline{(\sigma_{B,h})_k} \leftarrow N, N < h$ , but that do
              not come from a copy (see above). This way, if the current
              value of  $\sigma_{B,h}$  is starred, since its starless value will be
              discovered later on (or less starred values), thanks to the
              decreasing order on  $k$ , lines corresponding to what is called  $J$ 
              will necessarily be found at the next round, ending by  $\overline{(\sigma_{B,h})_{k-1}}$ 
              : the lines coming from the copy instruction above.      */

```

Il reste alors à lire R_s pour obtenir les couples d'indices des racines et des feuilles se correspondant. Pour ce faire, pour tout indice i sans drapeau de $Line(1)$, la ligne contenant l'ensemble des matrices décrivant des racines éventuelles de χ :

1. Identifier les cases de R_s débutant par i . Pour chacune :
 - (a) Accéder à la dernière valeur.
 - (b) Si elle ne porte pas de drapeau et qu'elle possède le même type que $leaf(\chi)$, alors c'est l'indice d'une feuille associée à la racine i .
 - (c) Sinon, si elle n'est pas du même type que $leaf(\chi)$, ne pas considérer (il s'agit d'un arc sortant de χ avant la racine).
 - (d) Sinon, il existe une ligne de R_s débutant par la valeur affectée du drapeau :
 - i. répéter sur cette ligne la procédure décrite ici depuis l'étape 1. L'algorithme s'arrête quand les parcours ne débouchent plus que sur des valeurs sans drapeau.



FOLIO ADMINISTRATIF

THESE DE L'UNIVERSITE DE LYON OPEREE AU SEIN DE L'INSA LYON

NOM : Barrellon
(avec précision du nom de jeune fille, le cas échéant)

DATE de SOUTENANCE : 27/11/2017

Prénoms : Vincent

TITRE : A Generic Approach towards the Collaborative Construction of Digital Scholarly Editions

NATURE : Doctorat

Numéro d'ordre : 2017LYSEI113

Ecole doctorale : Ecole doctorale d'Informatique et de Mathématiques de Lyon (EDA 512)

Spécialité : Informatique

RESUME : Digital Scholarly Editions are critically annotated patrimonial literary resources, in a digital form. Such editions roughly take the shape of a transcription of the original resources, augmented with critical information, that is, of structured data. In a collaborative setting, the structure of the data is explicitly defined in a schema, an interpretable document that governs the way editors annotate the original resources and guarantees they follow a common editorial policy. Digital editorial projects classically face two technical problems. The first has to do with the expressiveness of the annotation languages, that prevents from expressing some kinds of information. The second relies in the fact that, historically, schemas of long-running digital edition projects have to evolve during the lifespan of the project. However, amending a schema implies to update the structured data that has been produced, which is done either by hand, by means of ad-hoc scripts, or abandoned by lack of technical skills or human resources. In this work, we define the theoretical ground for an annotation system dedicated to scholarly edition. We define eAG, a stand-off annotation model based on a cyclic graph model, enabling the widest range of annotation. We define a novel schema language, SeAG, that permits to validate eAG documents on-the-fly, while they are being manufactured. We also define an inline markup syntax for eAG, reminiscent of the classic annotation languages like XML, but retaining the expressivity of eAG. Eventually, we propose a bidirectional algebra for eAG documents so that, when a SeAG S is amended, giving S' , an eAG I validated by S is semi-automatically translated into an eAG I' validated by S' , and so that any modification applied to I (resp. I') is semi-automatically propagated to I' (resp. I) – hence working as an assistance tool for the evolution of SeAG schemas and eAG annotations.

MOTS-CLÉS : Humanités numériques, Langage d'annotation, Schémas, Documents multistructurés, Transformations bidirectionnelles

Laboratoire (s) de recherche : Laboratoire d'informatique en image et systèmes d'information (LIRIS)

Directeur de thèse: Sylvie Calabretto

Président de jury :

Composition du jury :

Elisabeth Murisasco, Professeur des Universités, Université de Toulon
Ethan Munson, Professeur des Universités, University of Wisconsin-Milwaukee
Elena Pierazzo, Professeur des Universités, Université Grenoble Alpes
Jean-Yves Vion-Dury, Docteur, Naver Labs Europe
Sylvie Calabretto, Professeur des Universités, INSA Lyon
Pierre-Edouard Portier, Maître de Conférences, INSA Lyon
Olivier Ferret, Professeur des Universités, Université Lyon 2