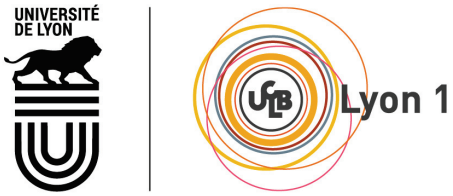
Algebras of Relations : from algorithms to formal proofs

Paul Brunet



N° d'ordre NNT : 2016LYSE1198

THÈSE DE DOCTORAT DE L'UNIVERSITÉ DE LYON
opérée au sein de
l'Université Claude Bernard Lyon 1

École Doctorale ED512
InfoMaths

Spécialité de doctorat : Informatique

Soutenue publiquement le 04/10/2016, par :
Paul Brunet

Algebras of Relations : From algorithms to formal proofs.

Après avis de :

Struth Georg, Professeur, University of Sheffield	Rapporteur
Kozen Dexter, Professeur, Cornell University	Rapporteur
Weil Pascal, Directeur de recherche CNRS, LaBRI	Rapporteur

Devant le jury composé de :

Struth Georg, Professeur, University of Sheffield	Examinateur
Muscholl Anca, Professeur, Université de Bordeaux	Examinatrice
Carton Olivier, Professeur, Université Paris Diderot	Examinateur
Schnoebelen Philippe, Directeur de recherche CNRS, LSV – ENS de Cachan	Examinateur
Silva Alexandra, Senior Lecturer, University College London	Examinatrice
Petrisan Daniela, Docteur, IRIF	Examinatrice
Malbos Philippe, Maître de Conférences, Université Claude Bernard Lyon 1	Examinateur
Pous Damien, Chargé de recherche CNRS, LIP – ENS de Lyon	Directeur de thèse

UNIVERSITE CLAUDE BERNARD - LYON 1

Président de l'Université	**M. le Professeur Frédéric FLEURY**
Président du Conseil Académique	M. le Professeur Hamda BEN HADID
Vice-président du Conseil d'Administration	M. le Professeur Didier REVEL
Vice-président du Conseil Formation et Vie Universitaire	M. le Professeur Philippe CHEVALIER
Vice-président de la Commission Recherche	M. Fabrice VALLÉE
Directeur Général des Services	M. Alain HELLEU

COMPOSANTES SANTE

Faculté de Médecine Lyon Est – Claude Bernard	Directeur : M. le Professeur J. ETIENNE
Faculté de Médecine et de Maïeutique Lyon Sud – Charles Mérieux	Directeur : Mme la Professeure C. BURILLON
Faculté d'Odontologie	Directeur : M. le Professeur D. BOURGEOIS
Institut des Sciences Pharmaceutiques et Biologiques	Directeur : Mme la Professeure C. VINCIGUERRA
Institut des Sciences et Techniques de la Réadaptation	Directeur : M. le Professeur Y. MATILLON
Département de formation et Centre de Recherche en Biologie Humaine	Directeur : Mme la Professeure A-M. SCHOTT

COMPOSANTES ET DEPARTEMENTS DE SCIENCES ET TECHNOLOGIE

Faculté des Sciences et Technologies	Directeur : M. F. DE MARCHI
Département Biologie	Directeur : M. le Professeur F. THEVENARD
Département Chimie Biochimie	Directeur : Mme C. FELIX
Département GEP	Directeur : M. Hassan HAMMOURI
Département Informatique	Directeur : M. le Professeur S. AKKOUCHE
Département Mathématiques	Directeur : M. le Professeur G. TOMANOV
Département Mécanique	Directeur : M. le Professeur H. BEN HADID
Département Physique	Directeur : M. le Professeur J-C PLENET
UFR Sciences et Techniques des Activités Physiques et Sportives	Directeur : M. Y.VANPOULLE
Observatoire des Sciences de l'Univers de Lyon	Directeur : M. B. GUIDERDONI
Polytech Lyon	Directeur : M. le Professeur E.PERRIN
Ecole Supérieure de Chimie Physique Electronique	Directeur : M. G. PIGNAULT
Institut Universitaire de Technologie de Lyon 1	Directeur : M. le Professeur C. VITON
Ecole Supérieure du Professorat et de l'Education	Directeur : M. le Professeur A. MOUGNIOTTE
Institut de Science Financière et d'Assurances	Directeur : M. N. LEBOISNE

Résumé

Les algèbres de relations apparaissent naturellement dans de nombreux cadres, en informatique comme en mathématiques. Elles constituent en particulier un formalisme tout à fait adapté à la sémantique des programmes impératifs. Les algèbres de Kleene constituent un point de départ : ces algèbres jouissent de résultats de décidabilité très satisfaisants, et admettent une axiomatisation complète. L'objectif de cette thèse a été d'étendre les résultats connus sur les algèbres de Kleene à des extensions de celles-ci.

Nous nous sommes tout d'abord intéressés à une extension connue : les algèbres de Kleene avec converse. La décidabilité de ces algèbres était déjà connue, mais l'algorithme prouvant ce résultat était trop compliqué pour être utilisé en pratique. Nous avons donné un algorithme plus simple, plus efficace, et dont la correction est plus facile à établir. Ceci nous a permis de placer ce problème dans la classe de complexité PSpace-complete.

Nous avons ensuite étudié les allégories de Kleene. Sur cette extension, peu de résultats étaient connus. En suivant des résultats sur des algèbres proches, nous avons établi l'équivalence du problème d'égalité dans les allégories de Kleene à l'égalité de certains ensembles de graphes. Nous avons ensuite développé un modèle d'automate original (les automates de Petri), basé sur les réseaux de Petri, et avons établi l'équivalence de notre problème original avec le problème de comparaison de ces automates. Nous avons enfin développé un algorithme pour effectuer cette comparaison dans le cadre restreint des treillis de Kleene sans identité. Cet algorithme utilise un espace exponentiel. Néanmoins, nous avons pu établir que la comparaison d'automates de Petri dans ce cas est ExpSpace-complète.

Enfin, nous nous sommes intéressés aux algèbres de Kleene Nominales. Nous avons réalisé que les descriptions existantes de ces algèbres n'étaient pas adaptées à la sémantique relationnelle des programmes. Nous les avons donc modifiées pour nos besoins, et ce faisant avons trouvé diverses variations naturelles de ce modèle. Nous avons donc étudié en détail et en Coq les ponts que l'on peut établir entre ces variantes, et entre le modèle "classique" et notre nouvelle version.

Abstract

Algebras of relations appear naturally in many contexts, in computer science as well as in mathematics. They constitute a framework well suited to the semantics of imperative programs. Kleene algebras are a starting point: these algebras enjoy very strong decidability properties, and a complete axiomatisation. The goal of this thesis was to export known results from Kleene algebra to some of its extensions.

We first considered a known extension: Kleene algebras with converse. Decidability of these algebras was already known, but the algorithm witnessing this result was too complicated to be practical. We proposed a simpler algorithm, which is more efficient, and whose correctness is easier to establish. It allowed us to prove that this problem lies in the complexity class PSpace-complete.

Then we studied Kleene allegories. Few results were known about this extension. Following results about closely related algebras, we established the equivalence between equality in Kleene allegories and the equality of certain sets of graphs. We then developed a new automaton model (so-called Petri automata), based on Petri nets. We proved the equivalence between the original problem and comparing these automata. In the restricted setting of identity-free Kleene lattices, we also provided an algorithm performing this comparison. This algorithm uses exponential space. However, we proved that the problem of comparing Petri automata lies in the class ExpSpace-complete.

Finally, we studied Nominal Kleene algebras. We realised that existing descriptions of these algebra were not suited to relational semantics of programming languages. We thus modified them accordingly, and doing so uncovered several natural variations of this model. We then studied formally the bridges one could build between these variations, and between the existing model and our new version of it. This study was conducted using the proof assistant Coq.

Remerciements

Merci Papa et Maman, de m'avoir soutenu, supporté, et de ne pas m'avoir (encore) renié. Merci Manon, Lilly et Simon, pour vos nombreuses délicates attentions. Merci Marguerite, sans toi ma vie serait moins facile et moins amusante. Merci Papy et Papy, je devrais vous appeler plus souvent...

Merci Damien, tu as été un directeur de thèse idéal (le mot n'est pas trop fort). Merci à mes camarades évadés avant moi, Val (Magnolias for ever), JM (qui est très gentil), Matthieu (on s'est croisé plusieurs fois au labo), Gaupy ("juste une bière alors"...), qui m'ont servi d'exemples, que dis-je, de modèles. Merci à tous les plumeux passés, présents et futurs, en particulier Colin, Anupam, Denis, Laure D (qui m'a amené des chocolats dans les moments les plus difficiles), Simon, Pierre (le père du GdT du vendredi soir), Patrick, Daniel. Merci aux assistantes du labo, et tout spécialement à Catherine qui a su gérer gentiment mon inaptitude totale à de nombreuses choses... Merci à Dominique et aux MILIP. Grâce à vous tous, ça a été un plaisir immense de passer ces 3 ans au LIP.

Merci à Sylvain, Manu, Romuald et Laure G, avec qui ça a été très agréable de donner des TD à Lyon 1. Merci aux thésards du LIP, longue vie aux JDD!

Merci aux faggots: Jésus, le grand singe et le vieil homme. Merci aux vieux lyonnais: p'tit Sam, Sabin, Inglorious Gorieux, Alvaro le magnifique. Merci Nico & Caro, merci Victor & Pauline, on a bien ri.

Merci Georg, Pascal et Dexter, d'avoir accepté le travail de rapporter cette thèse (et donc de la lire!). Je suis particulièrement reconnaissant pour les très nombreuses et pertinentes remarques que Georg a disséminé dans tout le manuscrit. Merci Alexandra, Daniela, Olivier, Philippe, Philippe, Anca et encore Georg de participer à mon jury de thèse. Je suis honoré d'avoir tous vos noms sur la première page!

Introduction

In many contexts in computer science and mathematics binary relations appear naturally. Indeed, a vast number of objects of interest either are relations, like orders or bisimulations, or can be seen as relations, like graphs or computer programs. A major benefit of relational approaches is the surprisingly small number of relational operators needed to express complex notions.

The question that has motivated us during this thesis was the computation of universal laws of relation algebra, that is to say checking whether or not some universally quantified equation over relations holds. This has wide ranging applications. Two of them seem of particular interest to the author:

Program verification: giving formal guarantees that computer programs do not have bugs is a major challenge of modern computer science. Because the programs we want to check are often millions of lines long, we need automated methods. One may encode programs and some of their properties as relational equations, and use the algebraic theory of relations to check whether a given program satisfies some property;

Automated reasoning: since the beginning of the twentieth century mathematicians have been increasingly concerned with the degree of confidence one could have with mathematical proofs. This concern is heightened when one deals with large proofs, as it is common place in computer science: proofs by case analysis often require long and tedious work. For these reasons automatic proof checkers and proof assistants have been developed, giving formal guarantees of correctness. In this context, it is of paramount importance to ease the use of these tools by providing powerful decision procedures, discharging the burden of proof to the machine.

Kleene Algebra, introduced by Stephen C. Kleene under the name "algebra of regular events [35]", provide an algebraic framework allowing to express properties of the operations of union ($\cup$), sequence or product ($\cdot$), and iteration or reflexive transitive closure (*), as well as the constants empty relation (0) and identity relation (1). It is well known that the corresponding equational theory is decidable, and that it is complete for both language and relation models.

As expressive as it may be, one may wish to integrate other usual operations in such a setting. Theories obtained this way, by addition of a finite set of equations to the axioms of Kleene Algebra, are called *extensions of Kleene Algebra*. In this thesis, we studied several such extensions.

Kleene algebra with converse The converse operator is of course very natural in relational theory. It exchanges the direction of a relation, thus allowing to express properties such as

- a relation R is symmetric: $R = R^\smile$;
- R is an equivalence relation: $R = (R \cup R^\smile)^\star$;
- R is a partial function: $R^\smile \cdot R \subseteq 1$.

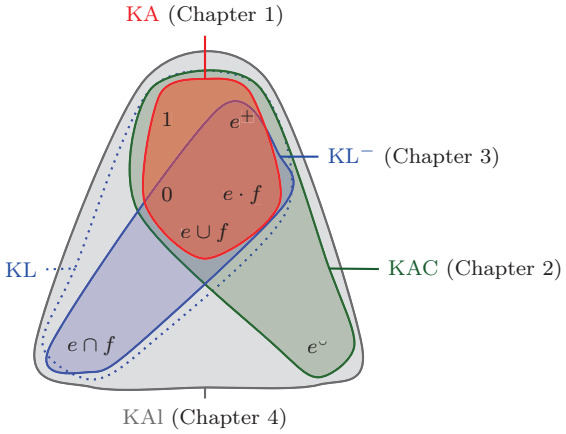


Figure 1: Overview

Bloom, Ésik, Stefanescu and Bernátsky [6, 26] gave a complete axiomatisation of this algebra, and proved that its equational theory is decidable. In Chapter 2 we reformulate some of their proofs, and give a new algorithm to compare expressions. We establish that the equivalence problem in Kleene algebra with converse is PSPACE-complete, and discuss time-efficient algorithms. This chapter is adapted from [18].

Kleene allegories and lattices Kleene lattices (KL) are Kleene algebra extended with an intersection operator. This operator is invaluable as it allows to express the conjunction of properties. When the converse is added, we get what we call Kleene allegories (KAl). Following work by Freyd and Scedrov [27], and by Andréka, Bredikhin, Mikulás and Németi [2, 3], we showed that the equivalence problem in KAl is equivalent to testing the equality of certain sets of graphs. We designed Petri-net based automata, called Petri automata, to recognise these sets of graphs. We obtained a Kleene theorem for them: for every expression over the signature of Kleene allegories one can produce an automaton recognising the same set of graphs, and vice versa. For the smaller signature of identity-free Kleene lattices (where the operations are restricted to 0, $\cdot$, $\cup$, $\cap$ and the non-zero iteration $^+$) we showed how to compare these automata. In Chapter 3 we present the automaton model and establish the Kleene theorem. In Chapter 4 we detail how this applies to Kleene allegories, and provide the comparison algorithm. Some of these results have been published in [16].

It is worth mentioning however that in Chapter 3 we use a rather different point of view, making everything about the definition and manipulation of sets of graphs. This stems from our realisation that what started out as a technical development actually produced an algebraic formalism well suited to discuss sets of series parallel graphs, and a corresponding operational framework to manipulate these sets.

Figure 1 summarises the algebras studied in the first four chapters.

Nominal Kleene algebra The last chapter goes in an rather different direction. The theory of nominal sets is getting a lot of attention these days, as it deals elegantly with crucial issues in computer science. Indeed it provides a purely algebraic and modular way of expressing properties of binders and variables in a vast array of models. It has been imported to Kleene algebra by Gabbay and Ciancia [28], and refined by Kozen, Silva, Petrisan and Mamouras [39, 40]. However, when trying to apply this to relational models, we ran into some problems that could only be solved by changing the formalism proposed by these authors. In Chapter 5 we investigate the relationship between the existing theory and several natural variations thereof. The goal here is to pave the way for a more extensive study of relational nominal Kleene algebras. This chapter is adapted from [17].

Contents

Introduction **v**

Contents **vii**

1 Preliminaries **1**
1.0 Notations and basic definitions 1
1.1 Algebra of relations 2
1.2 Regular languages and automata 3
1.3 Kleene Algebra 13

2 Algorithms for Kleene Algebra with Converse **17**
2.1 Introduction 17
2.2 Preliminary material 19
2.2.1 Languages with converse 19
2.2.2 Relations with converse: theory KAC 20
2.3 Confluence of the reduction relation 22
2.4 Closure of an automaton 23
2.4.1 Original construction 24
2.4.2 Intuitions 24
2.4.3 New construction 25
2.4.4 Example 29
2.5 Analysis and consequences 29
2.5.1 Relationship with Bloom et al's construction 29
2.5.2 Complexity 35
2.5.3 A polynomial-space algorithm 35
2.5.4 Time-efficient algorithms 36
2.6 Conclusion 39

3 A Kleene theorem for graph languages **41**
3.1 Introduction 41
3.2 Regular and recognisable sets of graphs 41
3.2.1 Graphs 41
3.2.2 Regular graph expressions 44
3.2.3 Petri automata 45
3.2.4 Building automata from expressions 48
3.3 Boxes 52
3.3.1 Categories of boxes 52
3.3.2 Templates 57
3.3.3 Atomic boxes and templates 58
3.4 Main theorem 59
3.4.1 A regular language of runs. 60
3.4.2 Computing the expression. 61
3.5 Relationship with Branching Automata 62
3.5.1 Definitions and Kleene Theorem 63
3.5.2 Comparison with Petri automata 64
3.6 Conclusion 66

4 Petri automata for Kleene Allegories **67**
4.1 Introduction . . . 67
4.2 Graphs and expressions . . . 68
4.3 From allegoric expressions to graph expressions . . . 71
4.4 Petri automata . . . 72
4.5 Comparing automata . . . 76
4.5.1 Restriction . . . 76
4.5.2 Intuitions . . . 76
4.5.3 Simulations . . . 78
4.5.4 The problems with converse and unit . . . 82
4.6 Complexity . . . 83
4.7 Relationship with standard Petri net notions . . . 85

5 A formal exploration of Nominal Kleene Algebra **87**
5.1 Introduction . . . 87
5.2 Expressions and proofs . . . 88
5.2.1 Atoms and letters . . . 88
5.2.2 Expressions and sets of expressions . . . 88
5.2.3 Proofs . . . 90
5.2.4 Theories . . . 91
5.3 Ordering theories . . . 94
5.3.1 Definitions . . . 94
5.3.2 Embeddings . . . 94
5.4 Relational interpretation of literate expressions . . . 97
5.5 Future work . . . 98

List of definitions **99**

Author's Contributions **101**

Bibliography **103**

First chapter

Preliminaries

"In the beginning was the Word."

— John, *The Holy Bible.*

In this section, we recall some well known results about regular languages and Kleene algebra.

1.0 Notations and basic definitions

Given a set X, we write $\mathcal{P}(X)$ for the powerset, the set of all subsets, of X, and $\mathcal{P}_f(X)$ for the set of finite subsets of X. The set of natural numbers is written $\mathbb{N}$. Composition of two functions f and g is written $f \circ g$; it maps x to $f(g(x))$. We write B^A or $A \to B$ for the set of functions from A to B. If $f \in X^B$ and $A \subseteq B$, then $f\restriction_A$ is the restriction of f to A. The set of partial functions from A to B is written $A \rightharpoonup B$. If f is a partial function, its domain, written $\mathrm{dom}(f)$, is the set of elements of A where f is defined. Let R be an equivalence relation over some set X and $x \in X$, then $[x]_R := \{y \in X \mid x\ R\ y\}$ is the equivalence class of x. The set $X/R := \{[x]_R \mid x \in R\}$ is the quotient of X by R.

1.0.1 Words and languages

An *alphabet* Σ is a finite set whose elements are called *letters*. We denote the *empty word* by ε, and the *set of all words* by $\Sigma^\star$. For any word w, $|w|$ is the *size* of w, meaning its number of letters; for any $1 \leqslant i \leqslant |w|$, we write $w(i)$ for the i^{th} letter of w and $w|_i := w(1)w(2)\cdots w(i)$ for its *prefix of size* i. The set of words of size strictly less than n is written $\Sigma^{<n}$. Given two words $u, v \in \Sigma^\star$, their *concatenation* is

$$uv := u(1)\cdots u(|u|)\, v(1)\cdots v(|v|).$$

The *set of all suffixes* of w is $suffixes(w) := \{v \mid \exists u : uv = w\}$. The *mirror image* of a word w is the word obtained by reversing the order of its letters: $w^\smile := w(|w|)\cdots w(1)$.

A *language* is a subset of $\Sigma^\star$. Given two languages $L, M \subseteq \Sigma^\star$, their *concatenation* is

$$L \cdot M := \{uv \mid u \in L \text{ and } v \in M\}.$$

The *mirror image* of L is the set of mirror images of words of L: $L^\smile := \{w^\smile \mid w \in L\}$. The n^{th} *power* of a language L is defined recursively as follows:

$$L^0 := \{\varepsilon\} \qquad\qquad L^{n+1} := L \cdot L^n.$$

The *star* of a language L may then be defined as:

$$L^\star := \bigcup_{n \in \mathbb{N}} L^n.$$

1.0.2 Binary relations

Given a set O, a *binary relation* over O is a subset of $O \times O$. We write $\mathcal{R}el\langle O\rangle$ for the set of binary relations over O. We denote by $R \cdot S$ the *composition* of two relations: $R \cdot S := \{\langle x, y\rangle \mid \exists z, x\ R\ z \text{ and } z\ R\ y\}$. The *identity relation* is defined as $\mathrm{Id}_O := \{\langle x, x\rangle \mid x \in O\}$. As for languages, the n^{th} *power* of a relation R is defined recursively as follows:

$$R^0 := \mathrm{Id}_O \qquad\qquad R^{n+1} := R \cdot R^n.$$

A relation R is called *reflexive* if $\mathrm{Id}_O \subseteq R$, and *transitive* if $R \cdot R \subseteq R$. The *reflexive-transitive closure* of R, denoted by $R^\star$, is the smallest relation containing R that is both reflexive and transitive. It may be explicitly expressed in two different yet equivalent ways:

$$R^\star = \bigcup_{n\in\mathbb{N}} R^n \qquad\qquad R^\star = \lim_{n\to+\infty} (\mathrm{Id}_O \cup R)^n .$$

(The second equation is very useful when O is finite: is provides an effective way of computing $R^\star$.)

Finally, for a set $E \subseteq O$ and a relation R over O, we write $E \cdot R$ for the *set of successors* of E by the R, i.e. $\{y \mid \exists x \in E, x\, R\, y\}$.

1.0.3 Algebraic structures

Definition 1.1 ((Commutative, Idempotent) Monoid)**.**
A *monoid* is a structure $\langle M, \cdot, 1\rangle$ satisfying the following laws:

$$x \cdot (y \cdot z) = (x \cdot y) \cdot z \qquad x \cdot 1 = x \qquad 1 \cdot x = x.$$

A monoid is called *commutative* (respectively *idempotent*) if it also satisfies the first (resp. second) law below.

$$x \cdot y = y \cdot x \qquad\qquad x \cdot x = x.$$

∗

Definition 1.2 (Group)**.**
A *group* is a structure $\left\langle G, \cdot, 1, \_^{-1}\right\rangle$ such that $\langle G, \cdot, 1\rangle$ is a monoid and the following laws hold:

$$x \cdot x^{-1} = 1 \qquad\qquad x^{-1} \cdot x = 1$$

∗

Definition 1.3 ((Idempotent) Semi-ring)**.**
A *semi-ring* is a structure $\langle S, \cdot, +, 1, 0\rangle$ such that $\langle S, \cdot, 1\rangle$ is a monoid, $\langle S, +, 0\rangle$ is a commutative monoid, and the following laws hold:

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c) \qquad (a + b) \cdot c = (a \cdot c) + (b \cdot c) \qquad 0 \cdot a = 0 \qquad a \cdot 0 = 0.$$

S is called *idempotent* if $\langle S, +, 0\rangle$ is idempotent. Such a semi-ring may be equipped with a pre-order relation $\leq$, defined by: $a \leq b \overset{\Delta}{\iff} a + b = b$. ∗

1.1 Algebra of relations

As we pointed out earlier, for every set O the algebra $\mathcal{R}el\,\langle O\rangle$ comes equipped with the operations of composition and reflexive transitive closure, as well as an identity relation. We may extend this signature with:

- the binary unions and intersections;
- the constants empty relation (0) and universal relation ($\top$);
- the unary reciprocal or converse ($^\smile$) and the transitive closure ($^+$);
- and many others, for instance the boolean complement, the residuals...

These algebras give rise to an (in)equational theory: a pair of terms e, f made from those operations and some variables $a, b, \ldots$ is a *valid (in)equation of the algebra of relations* if the corresponding (in)equality holds universally. Here are valid equations and inequations: they hold whatever the relations we assign to variables a, b, and c.

$$(a \cup b)^\star \cdot b \cdot (a \cup b)^\star \;=\; (a^\star \cdot b \cdot a^\star)^+ \tag{1.1}$$

$$a^\star \;\leq\; 1 \cup a \cdot a^\smile \cdot a^+ \tag{1.2}$$

$$a \cdot b \;\cap\; c \;\leq\; a \cdot (b \;\cap\; a^\smile \cdot c) \tag{1.3}$$

$$a^+ \;\cap\; 1 \;\leq\; (a \cdot a)^+ \tag{1.4}$$

In most of the thesis, we will investigate how to establish such laws by restricting the signature to more manageable fragments.

We generalise the notion of valid equation slightly to make it more formal. For a signature Θ, a *compatible class of models* is a family $\mathcal{X}$ of sets X equipped with all the operations in Θ (but possibly with more operations). For any X in $\mathcal{X}$ we get a notion of interpretation. Let Σ be a finite set of variables, and φ be a map from Σ to X. We can define an *interpretation function* $\widehat{\varphi}$ by extending φ to terms over the signature Θ using only variables from Σ. Let e and f be such terms, then $e = f$ is a *universal law for the class* $\mathcal{X}$, written $\mathcal{X} \models e = f$, if for every model X in $\mathcal{X}$ and every map $\varphi : \Sigma \to X$ we have $\widehat{\varphi}(e) = \widehat{\varphi}(f)$.

(This can be extended to inequalities if every model in the class $\mathcal{X}$ admits a partial ordering.)

Two such classes will be of particular interest to us:

- the class $\mathcal{R}el$ of binary relations, containing the algebra $\mathcal{R}el\,\langle O\rangle$ for every set O;
- the class $\mathcal{L}ang$ of languages, containing for every finite alphabet A the algebra $\mathcal{P}(A^\star)$.

Both classes are compatible with every signature we will use in this thesis.

1.2 Regular languages and automata

The theory of regular languages and finite state automata is a cornerstone of theoretical computer science. Most definitions and results have been around for decades, yet this line of research is still very much active. We include this section mainly to make our notations and definitions explicit. For the sake of completeness, we also include some proofs of classical results, especially when the intuitions behind these proofs are useful for the rest of this thesis. Extensive surveys of these notions include [22, 21, 56].

When dealing with languages, one is often interested in two things: specifying them and comparing them. Regular expressions provide a natural way of specifying languages in a compositional manner. On the other hand automata are very useful to compare languages, starting with the so-called "word problem": given a word u and a language L, is it the case that $u \in L$?

Let us fix for the remainder of this section an alphabet Σ.

1.2.1 Regular languages

Definition 1.4 (Regular expression)**.**
A *regular expression* is a term over the following syntax, where a denotes any letter from Σ:

$$e, f ::= 0 \;\mid\; 1 \;\mid\; a \;\mid\; e + f \;\mid\; e \cdot f \;\mid\; e^\star.$$

The set of regular expressions over Σ is written $\mathrm{Reg}\,\langle\Sigma\rangle$. The *size* of an expression e, written $|e|$, is the number of operators it uses (also: the number of inner nodes in its syntax tree). $*$

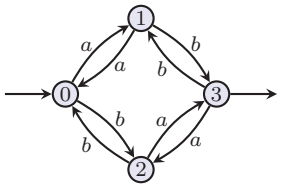


Figure 1.1: Deterministic automaton

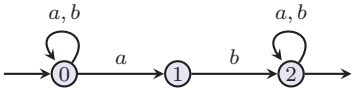


Figure 1.2: Non-deterministic automaton

These expressions can be interpreted as languages in a straightforward way:

Definition 1.5 (Regular languages)**.**
We define the function $[\![]\!] : \mathrm{Reg}\,\langle\Sigma\rangle \to \mathcal{P}\,(\Sigma^\star)$ by structural induction on regular expressions:

$$[\![a]\!] := \{a\} \qquad [\![0]\!] := \emptyset \qquad [\![1]\!] := \{\varepsilon\} \qquad [\![e^\star]\!] := [\![e]\!]^\star$$

$$[\![e + f]\!] := [\![e]\!] \cup [\![f]\!] \qquad [\![e \cdot f]\!] := [\![e]\!] \cdot [\![f]\!]\,.$$

A language L is called *regular* if there is an expression $e \in \mathrm{Reg}\,\langle\Sigma\rangle$ such that $L = [\![e]\!]$. ∗

1.2.2 Finite state automata

Elementary definitions and results

Definition 1.6 (Finite state automaton)**.**
A *deterministic automaton* is a tuple $\langle Q, \Sigma, q_0, F, \delta\rangle$; with Q a finite set of states, Σ an alphabet, $q_0 \in Q$ an initial state, $F \subseteq Q$ a set of final states and $\delta : Q \times \Sigma \to Q$ a transition function. A *non-deterministic automaton* is a tuple $\langle Q, \Sigma, I, F, \Delta\rangle$; with Q, Σ and F same as before, $I \subseteq Q$ a set of initial states and $\Delta \subseteq Q \times \Sigma \times Q$ a set of transitions. ∗

Two examples of automata are displayed in Figures 1.1 and 1.2.

As functions are a special case of relations, we will implicitly work with non-deterministic automata in the following, highlighting the differences only when relevant.

Remark. The definition we gave of a deterministic automaton is not completely standard: in many textbooks δ is considered to be a partial function (instead of a total one). In this setting our deterministic automata are called *complete* deterministic automata. However, this difference is very minor, as every partial function $\delta : Q \times \Sigma \rightharpoonup Q$ may be seen as a total function $\delta' : (Q \cup \{\bot\}) \times \Sigma \to (Q \cup \{\bot\})$.

Definition 1.7 (Relation induced by a letter, by a word)**.**
For any $a \in \Sigma$, the *relation induced on Q by a* is $\Delta(a) := \{\langle p, q\rangle \mid \langle p, a, q\rangle \in \Delta\}$. This can be extended to words by induction:

$$\widehat{\Delta}\,(\varepsilon) := \mathrm{Id}_Q \qquad \widehat{\Delta}\,(au) := \Delta\,(a) \cdot \widehat{\Delta}\,(u)\,. \qquad *$$

Definition 1.8 (Path)**.**
Let $\mathscr{A} = \langle Q, \Sigma, I, F, \Delta\rangle$ be an automaton, $p, q \in Q$ two states and $w \in \Sigma^\star$ a word. Let n be the length of u. A *path in $\mathscr{A}$ from p to q labelled by w* is a sequence of states $q_0, \dots q_n$ such that $q_0 = p$, $q_n = q$ and $\forall 0 \leqslant i < n, \langle q_i, w(i+1), q_{i+1}\rangle \in \Delta$.

We use the compact notation $p \xrightarrow{w}_{\mathscr{A}} q$ to denote the existence of such a path, or equivalently the fact that $p\ \widehat{\Delta}\,(w)\ q$. ∗

Notice that if $\mathscr{A}$ is deterministic, then for every state p and every word w there is exactly one state q such that $p \xrightarrow{w}_{\mathscr{A}} q$. Furthermore, the path from p to q labelled with w is unique. For these reasons, we will use the notation $q = \delta(p, w)$ in the following, thus implicitly extending the transition function δ to words.

Definition 1.9 (Language of an automaton, Recognisable language)**.**
Let $\mathscr{A} = \langle Q, \Sigma, I, F, \Delta \rangle$ be an automaton and $w \in \Sigma^\star$ a word. We say that w *is accepted by* $\mathscr{A}$ if there is a pair of states $\langle p, q \rangle \in I \times F$ such that $p \xrightarrow{w}_{\mathscr{A}} q$. The set of words accepted by $\mathscr{A}$, written $L(\mathscr{A})$, is called the *language recognised by the automaton* $\mathscr{A}$.

A language L is called *recognisable* if there is an automaton $\mathscr{A}$ such that $L = L(\mathscr{A})$. ∗

We may now state the following fundamental property of finite state automata:

Proposition 1.10 (Determinisation)**.** *A language is recognised by a non-deterministic automaton if and only if it is recognised by a deterministic automaton.* ■

Proof. As a deterministic automaton is a special case of a non-deterministic one, we just need to establish the "only if" part.

Let $\mathscr{A} = \langle Q, \Sigma, I, F, \Delta \rangle$ be an automaton. The determinised of $\mathscr{A}$ is defined as the automaton $\mathscr{A}^d := \langle \mathcal{P}(Q), \Sigma, I, F', \delta \rangle$ where:

$$F' := \{A \subseteq Q \mid A \cap F \neq \emptyset\} \qquad \delta := [A, a \mapsto A \cdot \Delta(a)].$$

It is a simple exercise to check that $L(\mathscr{A}) = L(\mathscr{A}')$. □

This construction is broadly known as the "powerset construction" for obvious reasons. It produces an automaton on exponential size, which is unavoidable, as shown later on in Example 1.15.

Definition 1.11 (Accessible state, accessible automaton)**.**
Let $\mathscr{A} = \langle Q, \Sigma, I, F, \Delta \rangle$ be an automaton. A state $q \in Q$ is called *accessible* if there exists a state $i \in I$ and a word $w \in \Sigma^\star$ such that $i \xrightarrow{w}_{\mathscr{A}} q$. We say that $\mathscr{A}$ is *accessible* if each of its states are accessible. ∗

Remark. It is very easy to modify the powerset construction to produce accessible deterministic automata: instead of taking as states $\mathcal{P}(Q)$, we only produce those we need. More precisely, we start from the initial state I (which is always accessible), then add $\delta(I, a)$ for every $a \in \Sigma$, and so on until we have a set of states stable under $\delta(\_, a)$.

In the following we will denote by $\mathscr{A}^d$ the accessible deterministic automaton obtained by the construction we sketched here.

Finite state automata are very versatile structures, and support a number of operations. This entails a number of closure properties of recognisable languages, as summarised in the following proposition.

Proposition 1.12 (Closure properties of recognisable languages.)**.** *Recognisable languages are closed under union, concatenation, star, intersection, mirror image and complementation.* ■

Proof. We give constructions performing each of the above operations. In each case the correctness of the construction being both easy and well known, we forgo the proof.

First, we describe the binary operations. Let $\mathscr{A}_1$ and $\mathscr{A}_2$ be two non-deterministic automata, with $\mathscr{A}_i = \langle Q_i, \Sigma, I_i, F_i, \Delta_i \rangle$, such that $Q_1 \cap Q_2 = \emptyset$[1].

union $\mathscr{A}^\cup := \langle Q_1 \cup Q_2, \Sigma, I_1 \cup I_2, F_1 \cup F_2, \Delta_1 \cup \Delta_2 \rangle$;

concatenation $\mathscr{A}^\cdot := \langle Q_1 \cup Q_2, \Sigma, I_1, F', \Delta_1 \cup \Delta_2 \cup \Delta' \rangle$, where:

$$F' := \begin{cases} F_1 \cup F_2 & \text{if } I_2 \cap F_2 \neq \emptyset \\ F_2 & \text{otherwise,} \end{cases} \qquad \Delta' := \left\{ \langle f, a, q \rangle \;\middle|\; \begin{array}{l} f \in F_1 \text{ and} \\ \exists i \in I_2 : \langle i, a, q \rangle \in \Delta_2 \end{array} \right\};$$

[1] This hypothesis is not restrictive as the language of an automaton is preserved by bijective renaming of the states.

intersection $\mathscr{A}^{\cap} := \langle Q_1 \times Q_2, \Sigma, I_1 \times I_2, F_1 \times F_2, \Delta' \rangle$, where:

$$\Delta' := \{ \langle \langle p_1, p_2 \rangle, a, \langle q_1, q_2 \rangle \rangle \mid \forall i \in \{1,2\}, \langle p_i, a, q_i \rangle \in \Delta_i \}.$$

Now for the unary operations, we first present the constructions for the star and mirror image, the complementation being slightly different. Let $\mathscr{A} = \langle Q, \Sigma, I, F, \Delta \rangle$ be a non-deterministic finite state automaton, and let $q_0 \notin Q$ be a fresh state.

star $\mathscr{A}^{\star} := \langle Q \cup \{q_0\}, \Sigma, \{q_0\}, \{q_0\}, \Delta \cup \Delta_i \cup \Delta_f \rangle$, where:

$$\Delta_i := \{ \langle q_0, a, q \rangle \mid \exists p \in I : \langle p, a, q \rangle \in \Delta \},$$
$$\Delta_f := \{ \langle p, a, q_0 \rangle \mid \exists q \in F : \langle p, a, q \rangle \in \Delta \cup \Delta_i \};$$

mirror $\mathscr{A}^{\smile} := \langle Q, \Sigma, F, I, \Delta' \rangle$ with $\Delta' := \{ \langle q, a, p \rangle \mid \langle p, a, q \rangle \in \Delta \}$.

For the complementation, we need to use a deterministic automaton. As stated in the previous lemma, this is not restrictive for recognisable languages. However, it does have a negative impact on the computational complexity of the construction, as the determinisation procedure is exponential. It is also worth mentioning that for this construction we really need the transition function δ to be total. Let $\mathscr{A} = \langle Q, \Sigma, q_0, F, \delta \rangle$ be a deterministic automaton. The complemented automaton is simply $\bar{\mathscr{A}} := \langle Q, \Sigma, q_0, Q \setminus F, \delta \rangle$. □

Minimal automaton An important result of automata theory is the existence of a unique minimal deterministic automaton for every recognisable language. In the following, the *size of an automaton* $\mathscr{A}$, written $|\mathscr{A}|$, is the cardinality of its set of states.

Proposition 1.13. *Let L be a recognisable language, there exists a deterministic automaton $\mathscr{A}^m$ such that $L(\mathscr{A}^m) = L$, and for every deterministic automaton $\mathscr{A}$ recognising L, either $|\mathscr{A}^m| < |\mathscr{A}|$ or $\mathscr{A}^m = \mathscr{A}$ (up-to bijective renaming of states).* ■

Proof. We begin by defining an equivalence relation $\sim_L$ on words:

$$u \sim_L v \overset{\Delta}{\iff} \forall w, (uw \in L \Leftrightarrow vw \in L).$$

Let $\mathscr{A} = \langle Q, \Sigma, I, F, \Delta \rangle$ be an automaton, we may define another equivalence relation $\sim_{\mathscr{A}}$ as follows:

$$u \sim_{\mathscr{A}} v \overset{\Delta}{\iff} \forall i \in I, \forall q \in Q, \left(i \xrightarrow{u}_{\mathscr{A}} q \Leftrightarrow i \xrightarrow{v}_{\mathscr{A}} q \right).$$

If $\mathscr{A}$ is accessible and deterministic with initial state q_0 and transition function δ, this equivalence relation may be reformulated as $\delta(q_0, u) = \delta(q_0, v)$. In this case, one can see that the equivalence classes of $\sim_{\mathscr{A}}$ are in bijection with Q (hence there are only finitely many of them).

Furthermore, suppose $L(\mathscr{A}) = L$. Then whenever $u \sim_{\mathscr{A}} v$, it must be the case that $u \sim_L v$. Indeed, we can prove this by contraposition: suppose there exists $w \in \Sigma^{\star}$ such that $uw \in L$ and $vw \notin L$. This means $\delta(q_0, uw) \in F$ but $\delta(q_0, vw) \notin F$, hence $\delta(q_0, uw) \neq \delta(q_0, vw)$. As $\delta(q_0, uw) = \delta(\delta(q_0, u), w)$ and $\delta(q_0, vw) = \delta(\delta(q_0, v), w)$ this entails $\delta(q_0, u) \neq \delta(q_0, v)$.

Let us now define explicitly $\mathscr{A}^m$ as $\langle \Sigma^{\star}/\sim_L, \Sigma, [\varepsilon]_{\sim_L}, F^m, \delta^m \rangle$, with $F^m := \Sigma^{\star}/\sim_L \cap \mathcal{P}(L)$, and $\delta^m([u]_{\sim_L}, a) := [ua]_{\sim_L}$. We immediately obtain that $L(\mathscr{A}^m) = L$ and that $\sim_{\mathscr{A}^m} = \sim_L$.

Let $\mathscr{A} = \langle Q, \Sigma, q_0, F, \delta \rangle$ be a deterministic automaton recognising L such that $|\mathscr{A}| \leqslant |\mathscr{A}^m|$. Remember that $\sim_{\mathscr{A}} \subseteq \sim_L$. This means that $\sim_{\mathscr{A}}$ has more equivalence classes that $\sim_L$. As we also know that the states of $\mathscr{A}$ are in bijection with $\Sigma^{\star}/\sim_{\mathscr{A}}$, this means that

$$|\mathscr{A}| \geqslant |\Sigma^{\star}/\sim_L| = |\mathscr{A}^m|.$$

Hence $\mathscr{A}$ and $\mathscr{A}^m$ have the same size. We also get $\sim_{\mathscr{A}} = \sim_L$, which allows us to conclude that $\mathscr{A}$ and $\mathscr{A}^m$ are equal up-to bijective renaming. □

The above proof follows Myhill-Nerode. It does not provide directly an algorithm to compute this minimal automaton, but several algorithms follow nonetheless the intuitions from this proof. However, there is a simpler and more surprising algorithm due to Brzozowski [19]. It relies on the following lemma:

Lemma 1.14. *If $\mathscr{A} = \langle Q, \Sigma, q_0, F, \delta\rangle$ is an accessible deterministic automaton, then $(\mathscr{A}^\smile)^d$ is the minimal automaton for the language $L(\mathscr{A})^\smile$.* ■

Proof. By unfolding the definitions, we get that $(\mathscr{A}^\smile)^d = \langle Q', \Sigma, q_0', F', \delta'\rangle$, where:

$$Q' \subseteq \mathcal{P}(Q), \qquad q_0' = F, \qquad F' = \{S \in Q' \mid q_0 \in S\},$$

$$\delta'(S,a) = \left\{ q \in Q \;\middle|\; \exists p \in S, p \xrightarrow{a}_{\mathscr{A}^\smile} q \right\}$$

We call this automaton $\mathscr{A}'$. From the definition of δ', we have for any word u and any state S of $\mathscr{A}'$:

$$\delta'(S,u) = \{q \in Q \mid \delta(q, u^\smile) \in S\}.$$

Let $L = L(\mathscr{A})$. To show that $\mathscr{A}'$ is minimal for $L^\smile$, we only need to check that $\sim_{L^\smile} \subseteq \sim_{\mathscr{A}'}$. Let $u, v \in \Sigma^\star$ such that $u \sim_{L^\smile} v$. By definition we have that for any word w, $uw \in L^\smile$ exactly when $vw \in L^\smile$. This may be reformulated as $w^\smile u^\smile \in L \Leftrightarrow w^\smile v^\smile \in L$, and finally as:

$$\forall w \in \Sigma^\star, wu^\smile \in L \Leftrightarrow wv^\smile \in L.$$

We need to check that $\delta'(q_0', u) = \delta'(q_0', v)$. We already know that:

$$\delta'(q_0', u) = \delta'(F, u) = \{q \in Q \mid \delta(q, u^\smile) \in F\}.$$

Because of the accessibility hypothesis, we can also devise a function $\omega : Q \to \Sigma^\star$ such that for any $q \in Q$, $\delta(q_0, \omega(q)) = q$. Hence we get that:

$$\delta(q, u^\smile) = \delta(\delta(q_0, \omega(q)), u^\smile) = \delta(q_0, \omega(q)\, u^\smile)$$

Which means that $\delta(q, u^\smile) \in F \Leftrightarrow \omega(q)\, u^\smile \in L$. Hence we have:

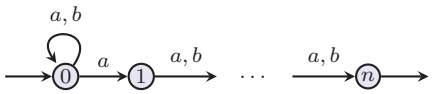
$$\begin{aligned}\delta'(q_0', u) &= \{q \in Q \mid \omega(q)\, u^\smile \in L\} \\ &= \{q \in Q \mid \omega(q)\, v^\smile \in L\} \\ &= \delta'(q_0', v).\end{aligned}$$ □

With this lemma, it is immediate to check that from any non-deterministic automaton $\mathscr{A}$ recognising the language L, the automaton $\left(\left((\mathscr{A}^\smile)^d\right)^\smile\right)^d$ is the minimal automaton for L.

Remark. A co-algebraic study of this construction was done by Bonchi et al [8].

The following example highlights the unavoidable exponential blowup when moving from non-deterministic to deterministic automata.

Example 1.15.
Consider the following automaton, using $n+1$ states:



This automaton recognises the language L denoted by the expression $(a+b)^\star a\,(a+b)^{n-1}$, whose size is also linear in n (as $(a+b)^n$ is a shorthand for an expression of size $4n-1$). However, the minimal automaton for this language has states $\{X \subseteq \{0,\dots,n\} \mid 0 \in X\}$ and transition function:

$$\delta(X,a) = \{0\} \cup \{i+1 \mid i \in X \setminus \{n\}\} \qquad \delta(X,b) = \{0\} \cup \{i+1 \mid i \in X \setminus \{0,n\}\}.$$

This automaton has 2^n states. We may see that it is minimal by realising that for any pair of words $u, v \in \Sigma^{<n}$ if $u \neq v$ then there is a word w such that $auw \in L \nLeftrightarrow avw \in L$:

- if $|u| > |v|$ then take $w = a^{n-1-|u|}$, and check that $auw \in L$ and $avw \notin L$;
- if $|u| = |v|$, we consider the first difference between the two and decompose them as $u = u_1 a u_2$ and $v = u_1 b u_2'$, with $|u_2| = |u_2'|$. Now take $w = a^{n-1-|u_2|}$, and check that $auw \in L$ and $avw \notin L$. •

Comparing automata The previous section already gives an algorithm to test whether two automata recognise the same language: one simply minimises both automata, and checks if the resulting automata are isomorphic (which is a simple enough operation). However, this approach is not very efficient, and doesn't seem to leave much room for optimisation. There is a range of methods to decide this problem, but we will only present here a quick overview of the bisimulation up-to family of algorithms.

First recall the notion of *simulation* [47]:

Definition 1.16 (Simulation)**.**
A relation S between the states of two automata $\mathscr{A}$ and $\mathscr{B}$ is a *simulation* if for all $p\ S\ q$ we have (a) if $p \xrightarrow{x}_{\mathscr{A}} p'$, then there exists a q' such that $q \xrightarrow{x}_{\mathscr{B}} q'$ and $p'\ S\ q'$, and (b) if $p \in F_{\mathscr{A}}$ then $q \in F_{\mathscr{B}}$. We say that $\mathscr{A}$ *is simulated by* $\mathscr{B}$ if there is a simulation S such that for any $p_0 \in I_{\mathscr{A}}$, there is $q_0 \in I_{\mathscr{B}}$ such that $p_0\ S\ q_0$. ∗

Definition 1.17 (Progress, Bisimulation, Bisimilarity)**.**
Let $\mathscr{A} = \langle Q, \Sigma, I, F, \Delta\rangle$ be an automaton, and $R, S \subseteq \mathcal{R}el\,\langle \mathcal{P}(Q)\rangle$ be two binary relations on sets of states. We say that R *progresses to* S, and write $R \rightarrowtail S$, if whenever $X\ R\ Y$ the following holds:

- $X \cap F \neq \emptyset$ if and only if $Y \cap F \neq \emptyset$.
- for every letter $a \in \Sigma$, $(X \cdot \Delta(a))\ S\ (Y \cdot \Delta(a))$.

A *bisimulation* is a relation R such that $R \rightarrowtail R$. Two sets of states $X, Y \subseteq Q$ are said to be *bisimilar* if there exists a bisimulation relating them. ∗

It is a simple exercise to check that bisimilarity is an equivalence relation, and is itself a bisimulation. The relevance of these notions is outlined by the following result:

Lemma 1.18. *Let $\mathscr{A}_1, \mathscr{A}_2$ be two automata with disjoint sets of states. For $i \in \{1,2\}$ we write $\mathscr{A}_i = \langle Q_i, \Sigma, I_i, F_i, \Delta_i\rangle$. Then $L(\mathscr{A}_1) = L(\mathscr{A}_2)$ if and only if I_1 and I_2 are bisimilar in the automaton $\mathscr{A}_1 \cup \mathscr{A}_2$.* ■

Proof. We begin by defining two functions $\delta_1, \delta_2 : \Sigma^\star \to \mathcal{P}(Q_1 \cup Q_2)$ by induction on words.

$$\delta_i(\varepsilon) := I_i$$
$$\delta_i(wa) := \delta_i(w) \cdot \Delta_i(a).$$

Notice that for every word $w \in \Sigma^\star$, we have $\delta_i(w) \subseteq Q_i$, and $w \in L(\mathscr{A}_i) \Leftrightarrow \delta_i(w) \cap F_i \neq \emptyset$.

Suppose $L(\mathscr{A}_1) = L(\mathscr{A}_2)$. We need to find a bisimulation relating I_1 and I_2. Let R be the following relation:

$$R := \{\langle \delta_1(w), \delta_2(w)\rangle \mid w \in \Sigma^\star\}.$$

As $\langle I_1, I_2\rangle = \langle \delta_1(\varepsilon), \delta_2(\varepsilon)\rangle$, we have $I_1 \ R \ I_2$. Suppose X and Y are related by R, then there must be some word $w \in \Sigma^\star$ such that $X = \delta_1(w)$ and $Y = \delta_2(w)$, which means $X \subseteq Q_1$ and $Y \subseteq Q_2$.

$$\begin{aligned} X \cap F \neq \emptyset &\Leftrightarrow X \cap F_1 \neq \emptyset \\ &\Leftrightarrow w \in L(\mathscr{A}_1) = L(\mathscr{A}_2) \\ &\Leftrightarrow Y \cap F_2 \neq \emptyset \Leftrightarrow Y \cap F \neq \emptyset. \end{aligned}$$

Furthermore, $\forall a \in \Sigma$, as $X \subseteq Q_1$, $X \cdot \Delta(a) = X \cdot \Delta_1(a) = \delta_1(wa)$ and for the same reason $Y \cdot \Delta(a) = \delta_2(wa)$, thus these two sets are related by R. Hence we have $R \rightarrowtail R$, so it is indeed a bisimulation.

Now suppose that we have a bisimulation $R \in \mathcal{R}el\langle \mathcal{P}(Q_1 \cup Q_2)\rangle$ such that $I_1 \ R \ I_2$. We will show that $L(\mathscr{A}_1) = L(\mathscr{A}_2)$ by induction on words, but we need a sightly more general statement to do so: for every word $w \in \Sigma^\star$, for every pair of sets of states $\langle X, Y\rangle \in \mathcal{P}(Q_1) \times \mathcal{P}(Q_2)$ such that $X \ R \ Y$, we have :

$$\exists \langle p_1, q_1\rangle \in X \times F_1 : p_1 \xrightarrow{w}_{\mathscr{A}_1} q_1 \Leftrightarrow \exists \langle p_2, q_2\rangle \in Y \times F_2 : p_2 \xrightarrow{w}_{\mathscr{A}_2} q_2.$$

For $w = \varepsilon$, this is equivalent to checking whether $X \cap F_1 \neq \emptyset \Leftrightarrow Y \cap F_2 \neq \emptyset$, which is enforced by the fact that R is a bisimulation. Now consider the case aw. The definition of bisimulation tell us that $X \cdot \Delta(a)$ and $Y \cdot \Delta(a)$ are again related by R. Because $X \subseteq Q_1$, we get that $X \cdot \Delta(a) = X \cdot \Delta_1(a) \subseteq Q_1$, and for the same reason $Y \cdot \Delta(a) \subseteq Q_2$. Hence the induction hypothesis applies to this pair, and we get:

$$\begin{aligned} &\exists \langle p_1, q_1\rangle \in X \times F_1 : p_1 \xrightarrow{aw}_{\mathscr{A}_1} q_1 \\ \Leftrightarrow &\exists \langle p_1, p_1', q_1\rangle \in X \times Q_1 \times F_1 : p_1 \xrightarrow{a}_{\mathscr{A}_1} p_1' \text{ and } p_1' \xrightarrow{w}_{\mathscr{A}_1} q_1 \\ \Leftrightarrow &\exists \langle p_1', q_1\rangle \in X \cdot \Delta_1(a) \times F_1 : p_1' \xrightarrow{w}_{\mathscr{A}_1} q_1 \\ \Leftrightarrow &\exists \langle p_2', q_2\rangle \in Y \cdot \Delta_2(a) \times F_2 : p_2' \xrightarrow{w}_{\mathscr{A}_2} q_2 \\ \Leftrightarrow &\exists \langle p_2, q_2\rangle \in Y \times F_2 : p_2 \xrightarrow{aw}_{\mathscr{A}_2} q_2. \end{aligned}$$

We conclude by noticing that $w \in L(A_i)$ if and only if $\exists \langle p_i, q_i\rangle \in I_i \times F_i : p_i \xrightarrow{w}_{\mathscr{A}_i} q_i$. □

It is then straightforward to devise an algorithm from this result:

1. start with a relation R containing only the pair $\langle I_1, I_2\rangle$,
2. choose a pair $\langle X, Y\rangle \in R$ that has not been processed yet,
3. if $X \cap F_1 = \emptyset$ and $Y \cap F_2 \neq \emptyset$ (or the converse) return `false`,
4. otherwise for every $a \in \Sigma$ enrich R with the pair $\langle X \cdot \Delta_1(a), Y(\Delta)_2(a)\rangle$
5. go to step 2.

The algorithm stops either when the relation R is a bisimulation, or when a counter example to language equality has been found.

The strength of the approach relies in its potential for optimisation, in particular via the so-called *up-to techniques* [58, 53]. A function $g : \mathcal{R}el\langle \mathcal{P}(Q)\rangle \to \mathcal{R}el\langle \mathcal{P}(Q)\rangle$ is called an up-to technique if every relation R such that $R \rightarrowtail g(R)$ is contained in a bisimulation. We discuss this in more details in Section 2.5.4.

Syntactic monoid Another important way of studying automata and recognisable languages is through monoids. We recommend the lecture notes by Pin [50] on this topic. We recall here a few definitions and basic results.

Definition 1.19 (Recognition by monoid)**.**
A monoid $\langle M, \cdot, 1\rangle$ *recognises* a language L if there is a subset $P \subseteq M$ and a monoid homomorphism $\varphi : \Sigma^\star \to M$ such that for every word $u \in \Sigma^\star$, $u \in L$ if and only if $\varphi(u) \in P$. Equivalently, P and φ should satisfy $L = \{u \in \Sigma^\star \mid \varphi(u) \in P\}$. ∗

Remark. It is straightforward to see that morphisms from $\Sigma^\star$ to M are exactly the functions obtained as follows: take any map f from Σ to M, and define the morphism $\widehat{f}$ inductively on words as:

$$\widehat{f}(\varepsilon) := 1 \qquad\qquad \widehat{f}(au) := f(a) \cdot \widehat{f}(u).$$

Given an automaton, the monoid of binary relations on states recognises the language of that automaton.

Definition 1.20 (Transition monoid)**.**
Let $\mathscr{A} = \langle Q, \Sigma, I, F, \Delta\rangle$ be an automaton. The *transition monoid* of $\mathscr{A}$ is

$$M_{\mathscr{A}} := \left\langle \left\{ \widehat{\Delta}(u) \;\middle|\; u \in \Sigma^\star \right\}, \cdot, \mathrm{Id}_Q \right\rangle. \qquad *$$

Lemma 1.21. *The transition monoid of $\mathscr{A}$ recognises the language of $\mathscr{A}$.* ■

Proof. Consider the function $\Delta : a \mapsto \Delta(a)$ and the subset P defined by

$$P := \{R \in M_{\mathscr{A}} \mid R \cap (I \times F) \neq \emptyset\}.$$

Clearly, $\widehat{\Delta}$ is a morphism from $\Sigma^\star$ to $M_{\mathscr{A}}$. Let u be any word. $\widehat{\Delta}(u) \in P$ means there exists $\langle i, f\rangle \in I \times F$ such that $i\ \widehat{\Delta}(u)\ f$, which in turn is equivalent to the existence of a path in $\mathscr{A}$ from i to f labelled with u. Hence we get $L(\mathscr{A}) = \left\{u \;\middle|\; \widehat{\Delta}(u) \in P\right\}$. □

Notice that the transition monoid of any automaton is finite. This prompts the following fundamental lemma:

Lemma 1.22. *A language is recognisable if and only if is is recognised by a finite monoid.* ■

Proof. Lemma 1.21 proves the "only if" direction. For the other direction, suppose we have a finite monoid $\langle M, \cdot, 1\rangle$, a map $\varphi : \Sigma \to M$ and a subset $P \subseteq M$. We build the following deterministic automaton:

$$\mathscr{A}_M := \langle M, \Sigma, 1, P, [x, a \mapsto x \cdot \varphi(a)]\rangle.$$

It is then straightforward to check that the language recognised by this automaton is exactly $\{u \mid \widehat{\varphi}(u) \in P\}$. □

As for deterministic automata, there is a unique minimal monoid recognising every recognisable language, called the syntactic monoid.

Definition 1.23 (Syntactic congruence, syntactic monoid)**.**
Let L be a recognisable language. The *syntactic congruence of L* is the following relation:

$$u \equiv_L v \overset{\Delta}{\iff} (\forall x, y \in \Sigma^\star, xuy \in L \Leftrightarrow xvy \in L).$$

The *syntactic monoid of L* is the quotient of $\langle \Sigma^\star, \cdot, \varepsilon\rangle$ by $\equiv_L$. We write it $\Sigma^\star / \equiv_L$. ∗

By construction, the syntactic congruence is an equivalence relation and a congruence.

Lemma 1.24. *The syntactic monoid recognises L.* ■

Proof. Let $\varphi : u \mapsto [u]_L$ be the function mapping any word to its equivalence class with respect to $\equiv_L$, and $P = \{[u]_L \mid u \in L\}$. We conclude by noticing that $u \equiv_L v$ and $v \in L$ entails $u \in L$, thus $u \in [v]_L \in P$ implies $u \in L$. □

Lemma 1.25. *A finite monoid $\langle M, \cdot, 1\rangle$ recognises L if and only if there exists a sub-monoid $M' \subseteq M$ and a surjective homomorphism $M' \to \Sigma^\star/\equiv_L$.* ■

Proof. Let $\langle M, \cdot, 1\rangle$ be a finite monoid, and φ and $P \subseteq M$ be such that $L = \{u \mid \widehat{\varphi}(u) \in P\}$. Consider the sub-monoid $M' := \{\varphi(u) \mid u \in \Sigma^\star\}$. This sub-monoid is generated by the images of letters in Σ by φ. Notice also that $\varphi(a) = \varphi(b) \Rightarrow a \equiv_L b$. Indeed if a and b are not syntactically congruent, there must be $x, y \in \Sigma^\star$ such that $xay \in L$ but $xby \notin L$ (or the symmetric). This entails $\widehat{\varphi}(x) \cdot \varphi(a) \cdot \widehat{\varphi}(y) \neq \widehat{\varphi}(x) \cdot \varphi(b) \cdot \widehat{\varphi}(y)$, hence $\varphi(a) \neq \varphi(b)$. This means that the map $\psi : \varphi(a) \mapsto [a]_L$ is well defined. It is then a simple exercise to check that this map yields a surjective homomorphism from M' to $\Sigma^\star/\equiv_L$.

On the other hand, if we have such a homomorphism ψ, then for every letter $a \in \Sigma$, we choose an element $\varphi(a) \in M'$ such that $\psi(\varphi(a)) = [a]_L$. From this definition we immediately get that $\psi(\widehat{\varphi}(u)) = [u]_L$. Hence we get $u \in L \Leftrightarrow \widehat{\varphi}(u) \in \{x \mid \exists u \in L : \psi(x) = [u]_L\}$. □

This lemma entails that the syntactic monoid is the smallest monoid recognising L (both in the sense of cardinality and of division ordering), and as such it is unique up to isomorphism.

Finally we have a simple way to build the syntactic monoid from the minimal automaton:

Lemma 1.26. *The syntactic monoid of a recognisable language is the transition monoid of its minimal automaton.* ■

Proof. Let $\mathscr{A}^m$ be a minimal automaton. Recall from the proof of Proposition 1.13 that the states of $\mathscr{A}^m$ are the equivalence classes of the relation $\sim_L$ defined as

$$u \sim_L v \overset{\Delta}{\Longleftrightarrow} \forall x \in \Sigma^\star, ux \in L \Leftrightarrow vx \in L.$$

First notice that $\forall q, \delta^m(q,u) = \delta^m(q,v)$ means that for every word x, $[xu]_{\sim_L} = [xv]_{\sim_L}$. Hence we get

$$\begin{aligned} &\forall q, \delta^m(q,u) = \delta^m(q,v) \\ \Leftrightarrow\ &\forall x, xu \sim_L xv \\ \Leftrightarrow\ &\forall x, y, xuy \in L \Leftrightarrow xvy \in L \\ \Leftrightarrow\ &u \equiv_L v. \end{aligned}$$

This proves (in non-deterministic notation) that $\left\{v \;\middle|\; \widehat{\Delta}(u) = \widehat{\Delta}(v)\right\} = [u]_L$ for every word u, thus proving that $M_{\mathscr{A}^m}$ and $\Sigma^\star/\equiv_L$ are isomorphic. □

1.2.3 Kleene theorem

The fundamental theorem of regular languages is the so-called Kleene theorem:

Theorem 1.27 (Kleene theorem)**.** *A language is regular if and only if it is recognisable.* ■

Proof (Sketch). To prove that regular languages are recognisable, we proceed by induction.

- The automaton $\mathscr{A}_1 = \langle\{\bullet\}, \Sigma, \{\bullet\}, \{\bullet\}, \emptyset\rangle$ recognises the language $\{\varepsilon\} = [\![1]\!]$.
- The automaton $\mathscr{A}_a = \langle\{\bullet, \circ\}, \Sigma, \{\bullet\}, \{\circ\}, \{\langle\bullet, a, \circ\rangle\}\rangle$ recognises $[\![a]\!]$.

- For every other operation, we may rely on Proposition 1.12 to conclude.

For the other direction, we need to introduce generalised automata, which are simply automata whose transitions are labelled with regular expressions instead of letters. To go from one state to another along a transition labelled with $e \in \mathrm{Reg}\langle\Sigma\rangle$ in such an automaton one has to read a word belonging to $[\![e]\!]$. It is easy to see every automaton as a generalised automaton. In this setting we may also restrict ourselves to automata with a single initial state $\bullet$ without incoming transitions and a single final state $\circ$ without outgoing transitions. Simply add two fresh states, and transitions labelled with 1 going from the first new state to the previously initial states, and from the previously final states to the other new state. We may further require that there is at most one transition between any pair of states: two transitions $\langle p, e, q\rangle$ and $\langle p, f, q\rangle$ may be merged as $\langle p, e + f, q\rangle$. Then the algorithm removes successively every state that is neither initial nor final, updating the set of transitions to preserve the language. To remove state q, for every pair of states p, r (with $p \neq q, \circ$ and $r \neq q, \bullet$), we replace the label of the transition between p and r by

$$\delta(p, r) + \delta(p, q)\left(\delta(q, q)\right)^{\star}\delta(q, r)$$

where $\delta(x, y)$ is either the label of the transition between x and y or 0 if there is no such transition. When every state except $\bullet$ and $\circ$ has been removed, we get an automaton of the shape

$$\langle\{\bullet, \circ\}, \mathrm{Reg}\langle\Sigma\rangle, \{\bullet\}, \{\circ\}, \{\langle\bullet, e, \circ\rangle\}\rangle.$$

By construction it recognises the same language than the initial automaton, and it is clear that the language it recognises is exactly $[\![e]\!]$. □

Remark. There exists a large number of methods to get an automaton from an expression, most of them more efficient than what we described here. In particular, one might be interested in:

- Glushkov's construction [30], as well as McNaughton and Yamada's [45], yield a non deterministic automaton whose states are the occurrences of letters in the expression;
- Thompson's construction [60], is related to the one presented here, but uses so-called ε-transitions to get a more compact automaton; a improvement on this construction was proposed by Ilie and Yu [33];
- Brzozowski [20] proposed yet another method, by introducing *derivatives of regular expressions*; this method was an inspiration for the more efficient construction by Antimirov [4].

Generally speaking, the non-deterministic automaton produced from a regular expression is linear in the size of the expression. On the other hand, the extraction of an expression from an automaton is an exponential operation. The algorithm we presented can be traced back to Kleene, and most of the existing algorithms for this problem follow the same method: the optimisations focus on choosing in which order states should be eliminated.

1.2.4 On space complexity

We will outline here a few results about the space complexity of some problems regarding regular expressions and automata. Let us begin with a simple technical lemma.

Lemma 1.28. *Let $\mathscr{A} = \langle Q, \Sigma, I, F, \Delta\rangle$ be an automaton. The language of $\mathscr{A}$ is not empty if and only if it contains a word of length smaller than $|Q|$.* ■

Proof. Let u be a word of minimum size accepted by $\mathscr{A}$. Let $q_0, \ldots, q_{|u|} \in Q$ an accepting run in $\mathscr{A}$ labelled by u:

$$\left\langle q_0, q_{|u|} \right\rangle \in I \times F; \qquad \forall 0 < i \leqslant |u|, \langle q_{i-1}, u(i), q_i \rangle \in \Delta.$$

If $|u| \geqslant |Q|$, then there must be $0 \leqslant i_1 < i_2 \leqslant |u|$ such that $q_{i_1} = q_{i_2}$. Hence we may build a run $(q'_i)_{0 \leqslant i \leqslant |u| - (i_2 - i_1)}$ as follows:

$$q'_i = \begin{cases} q_i \text{ for } i \leqslant i_1 \\ q_{i+(i_2-i_1)} \text{ otherwise.} \end{cases}$$

This run is accepting, hence the word $v = u(1) \ldots u(i_1)\, u(i_2+1) \ldots u(|u|)$ is accepted by $\mathscr{A}$, which breaks the assumption that u has minimum length among words in $L(\mathscr{A})$. □

Using this lemma, we get a upper-bound on the complexity of the comparison of two deterministic automata:

Proposition 1.29. *The problem of testing $L(\mathscr{A}) = L(\mathscr{B})$ for deterministic automata $\mathscr{A}$ and $\mathscr{B}$ lies in the complexity class* LogSpace. ■

Proof. First, notice that this problem is equivalent to testing whether $L(\mathscr{A}) \cap \overline{L(\mathscr{B})} = \emptyset$ and $\overline{L(\mathscr{A})} \cap L(\mathscr{B}) = \emptyset$. Looking at the proof of Proposition 1.12, we know that there is an automaton of size $|\mathscr{A}| \times |\mathscr{B}|$ recognising the language $L(\mathscr{A}) \cap \overline{L(\mathscr{B})}$. By Lemma 1.28 we know that this language is empty if and only if it does not contain a word of length smaller than $|\mathscr{A}| \times |\mathscr{B}|$.

The non-deterministic LogSpace algorithm will thus guess such a word one letter at a time, and simulate its reading in both automata. It only needs to store at any given time a counter of size $\log|\mathscr{A}| + \log|\mathscr{B}|$, a letter of constant size, and a state of logarithmic size in each automata. □

Because of the exponential blowup between non-deterministic and deterministic automata, this yields a PSpace upper-bound on the complexity of comparing arbitrary automata. Because the conversion between expressions and non-deterministic automata is linear this upper-bound may be transferred to the problem of comparing the languages of two regular expressions. Meyer and Stockmeyer [46] showed that this problem is actually PSpace-complete, by encoding the computation of a polynomial space Turing machine as regular expression. They also show in the same paper that adding a squaring operator e^2 to regular expressions yields a ExpSpace-complete comparison problem.

1.3 Kleene Algebra

Kleene algebra was introduced by Stephen C. Kleene [35], under the name *algebra of regular event*, as the equational theory of the set of regular languages. It provides an algebraic framework allowing one to express properties of the operations of union, sequence or product, and iteration, by studying the equivalence of expressions built with them.

The problem of giving an axiomatisation of this algebra was left open by Kleene, and has since been studied by several authors. Volodimir N. Redko [55] showed that any complete set of identities axiomatising this algebra must be infinite. For this reason, the axiomatisations that were proposed later on relied either on inference rules or on axiom schemes (i.e. finite presentations of an infinite set of identities).

Arto Salomaa [57] was the first to give a complete axiomatisation, that consisted in a dozen identities and an inference rule. However, this inference rule had a side condition which is not stable under substitution, hence this system is not appropriate to deal with Kleene algebra other than $\mathrm{Reg}\langle\Sigma\rangle$. Then, John H. Conway [22] proposed several other systems, introducing inference schemes (a set of inference rules indexed over the family of finite monoids). But

Conway could only postulate the completeness of these systems. Building on the systems of Salomaa and Conway, and using a remark by Maurice Boffa [7], Daniel Krob [41] proved during his PhD the completeness of a number of axiomatisations, including some of those proposed by Conway. In particular, he gave an axiomatisation without inference rules, using a dozen identities together with a set of identities indexed over finite simple groups. Zoltán Ésik [25] later simplified and generalised some of Krob's proofs. Independently of Krob's results, Dexter Kozen [36] proposed yet another axiomatisation, which he proved complete. This later proof relied on an algebraic presentation of the automaton minimisation algorithm, representing an automaton as a matrix over a Kleene algebra. This makes the technical development considerably more palatable than the proofs by Krob, as it follows a well-known algorithm using relatively simple objects. We will follow this presentation in this thesis.

Definition 1.30 (Kleene Algebra, KA)**.**
A *Kleene Algebra* is an algebraic structure $\langle K, +, \cdot, \_^\star, 0, 1\rangle$ such that $\langle K, +, \cdot, 0, 1\rangle$ is an idempotent semi-ring, and the operation $\_^\star$ satisfies the following axioms.

$$1 + a \cdot a^\star \leq a^\star \tag{1.5}$$
$$1 + a^\star \cdot a \leq a^\star \tag{1.6}$$
$$b + a \cdot x \leq x \Rightarrow a^\star \cdot b \leq x \tag{1.7}$$
$$b + x \cdot a \leq x \Rightarrow b \cdot a^\star \leq x. \tag{1.8}$$

The axiomatic theory KA consists in the axioms of an idempotent semi-ring together with axioms and implications (1.5) to (1.8). ∗

Kleene Algebras are thus the *models* of KA. We say that $e = f$ *is a logical consequence of the axioms of KA*, and write $\mathrm{KA} \vdash e = f$, if there is a proof of this equality using only the axioms of KA. Alternatively, $\mathrm{KA} \vdash \_ = \_$ is the smallest congruence over $\mathrm{Reg}\,\langle\Sigma\rangle$ satisfying the axioms of KA.

Algebras of languages and algebras of relations are Kleene algebras. Notice that in the case of languages, the interpretation defined by $\forall a \in \Sigma, \varphi(a) := \{a\}$ yields the regular language of the expression: $\widehat{\varphi}(e) = [\![e]\!]$. This interpretation is usually called the *standard interpretation* of an expression. We define the class KA of all Kleene algebras. It can easily be seen that the relations $\mathrm{KA} \models \_ = \_$ and $\mathrm{KA} \vdash \_ = \_$ coincide.

We now relate $\mathcal{L}ang \models e = f$ with $[\![e]\!] = [\![f]\!]$ for any regular expressions e and f.

Lemma 1.31. *For any pair* $e, f \in Reg\,\langle\Sigma\rangle$*,* $\mathcal{L}ang \models e = f$ *if and only if* $[\![e]\!] = [\![f]\!]$*.* ■

Proof. As $[\![\_]\!]$ is a particular language interpretation, the "only if" direction is obvious. The other direction follows easily from the equation:

$$\widehat{\varphi}(e) = \bigcup_{u \in [\![e]\!]} \widehat{\varphi}(u)\,. \tag{1.9}$$

This equation may be proved by structural induction on e, and entails that the equality of regular languages implies the equality of every language interpretation. □

We now do the same for relational interpretations.

Lemma 1.32. *For any pair* $e, f \in Reg\,\langle\Sigma\rangle$*,* $\mathcal{R}el \models e = f$ *if and only if* $[\![e]\!] = [\![f]\!]$*.* ■

Proof. For the "if" direction, as (1.9) holds also for relational interpretations, we may conclude by the same argument as before.

Now suppose that for every set O and every map $\varphi : \Sigma \to \mathcal{R}el\,\langle O\rangle$ we have $\widehat{\varphi}(e) = \widehat{\varphi}(f)$. We will choose a specific interpretation that will prove the equality of the corresponding

regular languages. Let $O = \Sigma^\star$ and $\varphi : a \mapsto \{\langle w, wa\rangle \mid w \in \Sigma^\star\}$. Notice that for any expression e and every pair of words u, v we have

$$\widehat{\varphi}(e) = \{\langle u, uv\rangle \mid u \in \Sigma^\star, v \in [\![e]\!]\}.$$

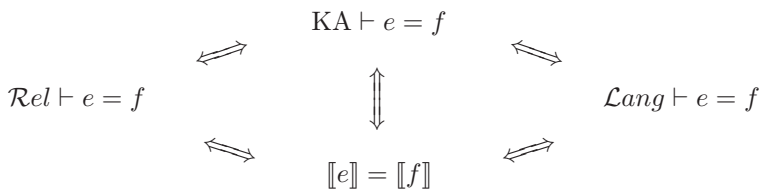
Then $u \in [\![e]\!]$ is equivalent to $\langle \varepsilon, u\rangle \in \widehat{\varphi}(e)$. Hence if $\widehat{\varphi}(e) = \widehat{\varphi}(f)$, we get $[\![e]\!] = [\![f]\!]$. □

(This last construction is the so-called *Cayley construction*, see [54].)

KA is *complete* with respect to the algebra of regular languages.

Theorem 1.33 ([36]). *For any $e, f \in Reg\langle\Sigma\rangle$, $KA \vdash e = f$ if and only if $[\![e]\!] = [\![f]\!]$.* ■

The proof of this theorem being very involved, we direct the interested reader to [36]. Put together, these results yield the fundamental theorem of Kleene algebra: for any pair of expressions $e, f \in \mathrm{Reg}\langle\Sigma\rangle$, we have:



This has several consequences:

- By Kleene's theorem the equality of two regular languages can be reduced to the equivalence of two finite automata, which is decidable. Hence, the theory KA is decidable, and PSPACE-complete.
- By Lemmas 1.31 and 1.32 this means that the equational theories of both relational and language Kleene algebra are also decidable.

Second chapter

Algorithms for Kleene Algebra with Converse

2.1 Introduction

The focus of this chapter is the extension of Kleene Algebra by an operation of *converse*. The converse of a word is its mirror image (the word obtained by reversing the order of the letters), and the converse $R^\smile$ of a relation R is its reciprocal ($x\, R^\smile\, y \overset{\Delta}{\iff} y\, R\, x$).

The question that arises once this theory is defined is its decidability: given two formal expressions built with the connectives product, sum, iteration and converse, can one decide automatically if they are equivalent, meaning that their equality can be proved using the axioms of the theory? Bloom, Ésik, Stefanescu and Bernátsky gave an affirmative answer to that question in two articles, [6] and [26], in 1995.

Although the algorithm they define proves the decidability result, it is too costly (in terms of time and memory consumption) to be used in concrete applications. In this chapter, beside some simplifications of the proofs given in [6], we give a new and more efficient algorithm to decide this problem, which we place in the complexity class PSpace.

Recall that for Kleene Algebra, we know of several axiomatisations, that are complete for both the language and relation models. The decidability of these theories can be reduced to the problem of comparing the languages of finite state automata.

Now let us add a unary operation of converse to regular expressions. We shall denote by $\mathrm{Reg}^\smile\langle X\rangle$ the set of regular expressions with converse over a finite alphabet X. While doing so, several questions arise:

1. Can the converse on languages and on relations be encoded with the same system of axioms?
2. What axioms do we need to add to KA to model these operations?
3. Are the resulting theories complete for languages and relations?
4. Are these theories decidable?

There is a simple answer to the first question: no. Indeed the equation $a \leq a \cdot a^\smile \cdot a$, which was proposed by Bloom Ésik and Stefanescu, is valid for any relation a (because if $\langle x, y\rangle \in a$, then $\langle x, y\rangle \in a$, $\langle y, x\rangle \in a^\smile$, and $\langle x, y\rangle \in a$, so that $\langle x, y\rangle \in a \cdot a^\smile \cdot a$). But this equation is not satisfied for all languages a (for instance, with the language $a = \{x\}$, $a \cdot a^\smile \cdot a = \{xxx\}$ and $x \notin \{xxx\}$). This means that there are two distinct theories corresponding to these two families of models. Let us begin by considering the case of languages.

Theorem 2.1 (Completeness of KAC$^-$ [6, Theorem 5.1])**.** *A complete axiomatisation of the class $\mathcal{L}ang$ of languages equipped with concatenation, union, star, and converse consists of the axioms of KA together with axioms* (2.1) *to* (2.4).

$$(a+b)^\smile = a^\smile + b^\smile \tag{2.1}$$
$$(a\cdot b)^\smile = b^\smile \cdot a^\smile \tag{2.2}$$
$$(a^\star)^\smile = (a^\smile)^\star \tag{2.3}$$
$$a^{\smile\smile} = a \tag{2.4}$$

We call this theory KAC$^-$; it is decidable. ■

As before, we write $\mathcal{L}ang \models e = f$ if e and f have the same language interpretations. To prove this result, one first associates to any expression $e \in \mathrm{Reg}^\smile \langle X \rangle$ an expression $\mathbf{e} \in \mathrm{Reg}\langle \mathbf{X} \rangle$, where $\mathbf{X}$ is an alphabet obtained by adding to X a disjoint copy of itself. Then, one proves that the following implications hold.

$$\mathcal{L}ang \models e = f \quad \Rightarrow \quad [\![\mathbf{e}]\!] = [\![\mathbf{f}]\!] \tag{2.5}$$

$$[\![\mathbf{e}]\!] = [\![\mathbf{f}]\!] \quad \Rightarrow \quad \mathrm{KAC}^- \vdash e = f \tag{2.6}$$

(That $\mathrm{KAC}^- \vdash e = f$ entails $\mathcal{L}ang \models e = f$ is obvious; decidability comes from that of regular languages equivalence.) We reformulate Bloom et al.'s proofs of these implications in elementary terms in Section 2.2.1.

As stated before, the equation $a \leq a \cdot a^\smile \cdot a$ provides a difference between languages with converse and relations with converse. It turns out that it is the only difference, in the sense that the following theorem holds:

Theorem 2.2 (Completeness and decidability of KAC)**.** *A complete axiomatisation of the class $\mathcal{R}el$ of relations equipped with composition, union, star, and converse consists of the axioms of KAC$^-$ together with the axiom* (2.7)*.*

$$a \leq a \cdot a^\smile \cdot a \tag{2.7}$$

We call this theory KAC; it is decidable. ■

(Completeness was established in [26, Theorem 1.1] and decidability in [6, Corollary 5.15].)

The proof of this result also relies on a translation into regular languages. Ésik et al. define a notion of *closure*, written $^\lhd$, for languages over $\mathbf{X}$, and they prove the following implications:

$$\mathcal{R}el \models e = f \quad \Rightarrow \quad {}^\lhd[\![\mathbf{e}]\!] = {}^\lhd[\![\mathbf{f}]\!] \tag{2.8}$$

$${}^\lhd[\![\mathbf{e}]\!] = {}^\lhd[\![\mathbf{f}]\!] \quad \Rightarrow \quad \mathrm{KAC} \vdash e = f \tag{2.9}$$

(Again, that $\mathrm{KAC} \vdash e = f$ entails $\mathcal{R}el \models e = f$ is obvious.) The first implication (2.8) was proven in [6]; we give a new formulation of this proof in Section 2.2.2. The second implication (2.9) was proved in [26].

The last consideration is the decidability of KAC. To this end, Bloom et al. propose a construction to obtain an automaton recognising ${}^\lhd L$, when given an automaton recognising L. Decidability follows: to decide whether $\mathrm{KAC} \vdash e = f$ one can build two automata recognising ${}^\lhd[\![\mathbf{e}]\!]$ and ${}^\lhd[\![\mathbf{f}]\!]$ and check if they are equivalent. Unfortunately, their construction tends to produce huge automata, which makes it inappropriate for practical applications. We propose a new and simpler construction in Section 2.4, which we analyse in Section 2.5:

- we compare it to Bloom et al.'s construction by exhibiting a bisimulation relation and by showing how our construction makes it possible to share more states (Section 2.5.1);
- we give a simple bound on the size of the produced automata (Section 2.5.2);
- we use this bound to provide a PSPACE algorithm, to deduce that the problem of equivalence in KAC is PSPACE-complete (Section 2.5.3);
- we finally provide algorithms that are not PSPACE but time-efficient in practice, using "up to techniques" [9, 53] for bisimulations (Section 2.5.4).

2.2 Preliminary material

2.2.1 Languages with converse: theory KAC$^-$

The following construction is due to Ésik et al., here we merely reformulate it slightly.

We consider regular expressions with converse over a finite alphabet X. The alphabet $\mathbf{X}$ is defined as $X \cup X'$, where $X' := \{x' \mid x \in X\}$ is a disjoint copy of X. As a shorthand, we use $'$ as an internal operation on $\mathbf{X}$ going from X to X' and from X' to X such that if $x \in X$, $x' := x' \in X'$ and $(x')' := x \in X$. An important operation in the following is the translation of an expression $e \in \mathrm{Reg}^{\smile}\langle X\rangle$ to an expression $\mathbf{e} \in \mathrm{Reg}\langle \mathbf{X}\rangle$. We proceed to its definition in two steps.

Let $\tau(e)$ denote the normal form of an expression $e \in \mathrm{Reg}^{\smile}\langle X\rangle$ in the following convergent term rewriting system:

$$\begin{array}{ccc} (a+b)^{\smile} \to a^{\smile} + b^{\smile} & 0^{\smile} \to 0 & (a^{\star})^{\smile} \to (a^{\smile})^{\star} \\ (a \cdot b)^{\smile} \to b^{\smile} \cdot a^{\smile} & 1^{\smile} \to 1 & a^{\smile\smile} \to a \end{array}$$

The corresponding equations being derivable in KAC$^-$, one easily obtains that:

$$\forall e \in \mathrm{Reg}^{\smile}\langle X\rangle\,,\ \mathrm{KAC}^- \vdash \tau(e) = e \tag{2.10}$$

We finally denote by $\mathbf{e}$ the expression obtained by further applying the substitution ν defined as $[x^{\smile} \mapsto x', (\forall x \in \mathbf{X})]$, i.e., $\mathbf{e} := \nu(\tau(e))$. (Note that $\mathbf{e} \in \mathrm{Reg}\langle \mathbf{X}\rangle$: it is regular, all occurrences of the converse operation having been eliminated.)

As explained in the introduction, Bloom et al.'s proof [6] amounts to proving the implications (2.5) and (2.6). It is worth noticing that this proof is rather simple, especially when compared to other completeness proofs. Completeness is obtained relatively to Kleene Algebra, which allows one to use any complete axiomatisation of KA as a basis to build an axiomatisation of KAC$^-$. We include a syntactic and elementary presentation of this proof, for the sake of completeness.

Lemma 2.3. *For all $e, f \in Reg^{\smile}\langle X\rangle$, $\mathcal{L}ang \models e = f$ entails $[\![\mathbf{e}]\!] = [\![\mathbf{f}]\!]$.* ■

Proof. Let us write $X_{\bullet} := X \uplus \{\bullet\}$ and consider the following interpretations (which appear in [6, proof of Proposition 4.3]):

$$\begin{array}{rl} \mu : X \longrightarrow \mathcal{P}\left(X_{\bullet}^{\star}\right) & \eta : \mathbf{X} \longrightarrow \mathcal{P}\left(X_{\bullet}^{\star}\right) \\ x \longmapsto \{x \cdot \bullet\} & x \in X \longmapsto \{x \cdot \bullet\} \\ & x' \in X' \longmapsto \{\bullet \cdot x\} \end{array}$$

One can check that $\widehat{\eta}$ is injective modulo equality of denoted languages, in the sense that for any expression $e \in \mathrm{Reg}\langle \mathbf{X}\rangle$, we have

$$\widehat{\eta}(e) = \widehat{\eta}(f) \text{ implies that } [\![e]\!] = [\![f]\!]\ . \tag{2.11}$$

By a simple induction on e, we get $\widehat{\mu}(\tau(e)) = \widehat{\eta}(\nu(\tau(e))) = \widehat{\eta}(\mathbf{e})$. Using (2.10), we further obtain that $\mathcal{L}ang \models \tau(e) = e$. We thus deduce that $\widehat{\mu}(e) = \widehat{\eta}(\mathbf{e})$. All in all, we obtain:

$$\mathcal{L}ang \models e = f \Rightarrow \widehat{\mu}(e) = \widehat{\mu}(f) \Rightarrow \widehat{\eta}(\mathbf{e}) = \widehat{\eta}(\mathbf{f}) \Rightarrow [\![\mathbf{e}]\!] = [\![\mathbf{f}]\!]\,. \qquad \square$$

The second implication is even more immediate, using the completeness of KA with respect to language equivalence.

Lemma 2.4. *For all $e, f \in Reg^{\smile}\langle X\rangle$, if $[\![\mathbf{e}]\!] = [\![\mathbf{f}]\!]$ then KAC$^-$ $\vdash e = f$.* ■

Proof. By completeness of KA [41, 36], if $[\![\mathbf{e}]\!] = [\![\mathbf{f}]\!]$, then we know that there is a proof π_1 of KA $\vdash \mathbf{e} = \mathbf{f}$. As the axioms of KA are contained in those of KAC$^-$, the same proof can be seen as a proof of KAC$^-$ $\vdash \mathbf{e} = \mathbf{f}$. By substituting x' by $(x^\smile)$ everywhere in this proof, we get a new proof π_2 of KAC$^-$ $\vdash \tau(e) = \tau(f)$. By (2.10) and transitivity we thus get KAC$^-$ $\vdash e = f$. □

We finally deduce the following theorem:

Theorem 2.5. *$\mathcal{L}ang \models e = f \Leftrightarrow [\![\mathbf{e}]\!] = [\![\mathbf{f}]\!] \Leftrightarrow KAC^- \vdash e = f$.* ■

Since the regular expressions $\mathbf{e}$ and $\mathbf{f}$ can be computed easily from e and f, the problem of equivalence in KAC$^-$ thus reduces to an equality of regular languages, which makes it decidable.

2.2.2 Relations with converse: theory KAC

We now move to the equational theory generated by relational models. It turns out that this theory is characterised using "closed" languages on the extended alphabet $\mathbf{X}$. To define this closure operation, we first define a mirror operation $\overline{w}$ on words over $\mathbf{X}$, such that $\overline{\varepsilon} := \varepsilon$ and for any $x \in \mathbf{X}$ and $w \in \mathbf{X}^\star$, $\overline{wx} := x'\overline{w}$. Accordingly with the axiom (2.7) of KAC we define a reduction relation $\rhd$ on words over $\mathbf{X}$ by the following word rewriting rule:

$$w\overline{w}w \rhd w \tag{2.12}$$

More formally:

Definition 2.6 (Reduction relation)**.**
Let u, v be two words over $\mathbf{X}$, *u reduces to v*, written $u \rhd v$, if there are words $u_1, u_2, w \in \mathbf{X}^\star$ such that: $u = u_1 w\overline{w}wu_2$ and $v = u_1 wu_2$.

We call $w\overline{w}w$ a *pattern* of *root* w. The *last two thirds* of the pattern are $\overline{w}w$. ∗

Following [6, 26], we extend this relation into a closure operation on languages.

Definition 2.7 (Closure by $\rhd$)**.**
The *closure* of a language $L \subseteq \mathbf{X}^\star$ is the smallest language containing L that is downward-closed with respect to $\rhd$:

$$^\lhd L := \{v \mid \exists u \in L : u \rhd^\star v\} \quad .$$

(As usual $\rhd^\star$ is the reflexive transitive closure of the relation $\rhd$.) ∗

Example 2.8.
If $X = \{a, b, c, d\}$, then $\mathbf{X} = \{a, b, c, d, a', b', c', d'\}$, and $\overline{ab'} = ba'$. We have the reduction $cab'ba'ab'd' \rhd cab'd'$, by triggering a pattern of root ab'. For $L = \{aa'a, b, cab'ba'ab'd'\}$, we have $^\lhd L = L \cup \{a, cab'd'\}$. •

Now we define a family of languages which plays a prominent role in the sequel.

Definition 2.9 (Language $\Gamma(w)$)**.**
For any word $w \in \mathbf{X}^\star$, we define inductively a regular language $\Gamma(w)$ by:

$$\Gamma(\varepsilon) := \{\varepsilon\} \qquad \forall x \in \mathbf{X}, \forall w \in \mathbf{X}^\star,\ \Gamma(wx) := (\{x'\}\,\Gamma(w)\,\{x\})^\star\,. \qquad ∗$$

An equivalent operator called G is used in [6]: we actually have $\Gamma(w) = G(\overline{w})$, and our recursive definition directly corresponds to [6, Proposition 5.11.(2)]. By using such a simple recursive definition, we avoid the need for the notion of *admissible maps*, which is extensively used in [6].

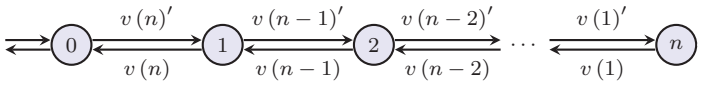


Figure 2.1: Automaton $\mathscr{G}(v)$ recognising $\Gamma(v)$, with $|v| = n$.

Instead, we just have the following property to establish, which illustrates why these languages are of interest: if u is a word in $\Gamma(w)$, then wu reduces to w. More precisely, w can be decomposed as $w = vt$ and u reduces to $\bar{t}t$. Therefore, in the context of recognition by an automaton, $\Gamma(w)$ contains all the words that could potentially be skipped after reading w, in an automaton recognising a language closed by $\rhd$.

Proposition 2.10. *For all words u and v, $u \in \Gamma(v) \Leftrightarrow \exists t \in suffixes(v) : u \rhd^\star \bar{t}t$.* ■

Proof. As the proof of the implication from left to right is routine but a bit lengthy, we omit it here.

For the converse implication, we first define the language: $\Gamma'(v) := \{\bar{t}t \mid t \in suffixes(v)\}$. We thus have to show that the upward closure of $\Gamma'(v)$ is contained in $\Gamma(v)$. We first check that this language satisfies $\Gamma'(\varepsilon) = \varepsilon$ and $\Gamma'(vx) = \varepsilon + x'\Gamma'(v)x$, which allows us to deduce that $\Gamma'(v) \subseteq \Gamma(v)$ by a straightforward induction.

It thus suffices to show that $\Gamma(v)$ is upward-closed with respect to $\rhd$. For this, we introduce the family of automata $\mathscr{G}(v)$ depicted in Figure 2.1. One can check that $\mathscr{G}(v)$ recognises $\Gamma(v)$ by a simple induction on v. One can moreover notice that in this automaton, if $p \xrightarrow{x}_{\mathscr{G}(v)} q$, then $q \xrightarrow{x'}_{\mathscr{G}(v)} p$. More generally, for any word u, if $p \xrightarrow{u}_{\mathscr{G}(v)} q$, then $q \xrightarrow{\overline{u}}_{\mathscr{G}(v)} p$. So if $u_1wu_2 \in \Gamma(v)$, then by definition of the automaton we have $0 \xrightarrow{u_1}_{\mathscr{G}(v)} q_1 \xrightarrow{w}_{\mathscr{G}(v)} q_2 \xrightarrow{u_2}_{\mathscr{G}(v)} 0$, and thus, by the previous remark:

$$0 \xrightarrow[\mathscr{G}(v)]{u_1} q_1 \xrightarrow[\mathscr{G}(v)]{w} q_2 \xrightarrow[\mathscr{G}(v)]{\overline{w}} q_1 \xrightarrow[\mathscr{G}(v)]{w} q_2 \xrightarrow[\mathscr{G}(v)]{u_2} 0 \quad ,$$

i.e., $u_1w\overline{w}wu_2 \in \Gamma(v)$. In other words, for any words v and w and any $u \in \Gamma(v)$, if $w \rhd u$ then w is also in $\Gamma(v)$, meaning exactly that $\Gamma(v)$ is upward-closed with respect to $\rhd$.

Since $\Gamma'(v) \subseteq \Gamma(v)$, we deduce that $\Gamma(v)$ contains the upward closure of $\Gamma'(v)$, as expected. □

We now have enough material to embark in the proof of the implication (2.8) from the introduction:

Lemma 2.11. *If two expressions $e, f \in Reg^{\smile}\langle X\rangle$ are equal for all interpretations in all relational models, then ${}^{\lhd}[\![\mathbf{e}]\!] = {}^{\lhd}[\![\mathbf{f}]\!]$.* ■

Proof. Bloom et al. [6] consider specific relational interpretations: for any word $u \in \mathbf{X}^\star$ and for any letter $x \in \mathbf{X}$, they define

$$\varphi_u(x) := \{\langle i-1, i\rangle \mid u(i) = x\} \cup \{\langle i, i-1\rangle \mid u(i) = x'\} \subseteq \{0, \ldots, n\}^2 \quad ,$$

where $n := |u|$. The key property of those interpretations is the following:

$$\langle 0, n\rangle \in \widehat{\varphi_u}(v) \Leftrightarrow v \rhd^\star u \quad . \tag{2.13}$$

We give a new proof of this property, by using the automaton $\Phi(u)$ depicted in Figure 2.2. By definition of $\Phi(u)$ and φ_u, we have that

$$(i, j) \in \varphi_u(x) \Leftrightarrow i \xrightarrow{x}_{\Phi(u)} j \quad .$$

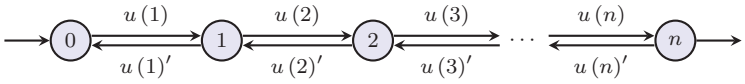


Figure 2.2: Automaton $\Phi(u)$, with $|u| = n$.

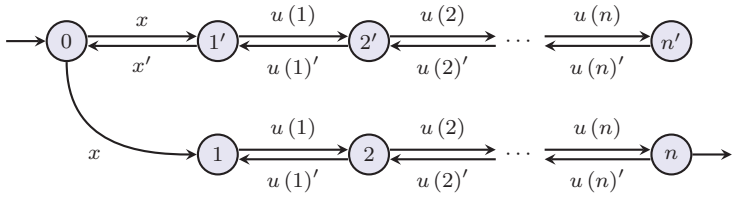


Figure 2.3: Automaton $\Phi'(xu)$, with $|u| = n$, language equivalent to $\Phi(xu)$.

Therefore, proving (2.13) amounts to proving

$$v \in L(\Phi(u)) \Leftrightarrow v \rhd^\star u \ . \tag{2.14}$$

First notice that $i \xrightarrow{x}_{\Phi(u)} j \Leftrightarrow j \xrightarrow{x'}_{\Phi(u)} i$. We extend this to paths (as in the proof of Proposition 2.10) and then prove that if $s \rhd t$ and $i \xrightarrow{t}_{\Phi(u)} j$ then $i \xrightarrow{s}_{\Phi(u)} j$. As u is clearly in $L(\Phi(u))$, any v such that $v \rhd^\star u$ is also in $L(\Phi(u))$.

We proceed by induction on u for the other implication. The case $u = \varepsilon$ being trivial, we consider $v \in L(\Phi(xu))$. We introduce a second automaton $\Phi'(xu)$ given in Figure 2.3, that recognises the same language as $\Phi(xu)$. The upper part of this automaton is actually the automaton $\mathscr{G}(\overline{xu})$ (as given in Figure 2.1), recognising the language $\Gamma(\overline{xu})$. Moreover, the lower part starting from state 1 is the automaton $\Phi(u)$. This allows us to obtain that $L(\Phi(xu)) = \Gamma(\overline{xu})xL(\Phi(u))$. Hence, for any $v \in L(\Phi(xu))$, there are $v_1 \in \Gamma(\overline{xu})$ and $v_2 \in L(\Phi(u))$ such that $v = v_1 x v_2$. By induction, we get $v_2 \rhd^\star u$, and by Proposition 2.10 we know that $v_1 \rhd^\star \overline{w}w$, with $w \in suffixes(\overline{xu})$. That means that $\overline{xu} = tw$, for some word t, so $xu = \overline{tw} = \overline{w}\ \overline{t}$. If we put everything back together:

$$v = v_1 x v_2 \rhd^\star v_1 x u \rhd^\star \overline{w}wxu = \overline{w}w\overline{w}\ \overline{t} \rhd \overline{w}\ \overline{t} = xu \ .$$

This concludes the proof of (2.14), and thus (2.13).

We follow Bloom et al.'s proof [6] to deduce that the implication (2.8) from the introduction holds: we first prove that for all $e \in \mathrm{Reg}\,\langle \mathbf{X} \rangle$, we have

$$\begin{aligned} u \in {}^{\lhd}[\![e]\!] &\Leftrightarrow \exists v \in [\![e]\!]\,, v \rhd^\star u && \text{(by definition)} \\ &\Leftrightarrow \exists v \in [\![e]\!]\,, \langle 0, n\rangle \in \widehat{\varphi_u}(v) && \text{(by (2.13))} \\ &\Leftrightarrow \langle 0, n\rangle \in \widehat{\varphi_u}(e) \ . \end{aligned}$$

(For the last line, we use the fact that for any relational interpretation φ, we have $\widehat{\varphi}(e) = \bigcup_{v \in [\![e]\!]} \widehat{\varphi}(v)$.)

Furthermore, as $\varphi_u(x') = \varphi_u(x)^\smile$, we can prove that $\widehat{\varphi}_u(e) = \widehat{\varphi}_u(\mathbf{e})$. Therefore, for all expressions $e, f \in \mathrm{Reg}^\smile\,\langle X \rangle$ such that $\mathcal{R}el \models e = f$, we have $\widehat{\varphi_u}(\mathbf{e}) = \widehat{\varphi_u}(e) = \widehat{\varphi_u}(f) = \widehat{\varphi_u}(\mathbf{f})$, and we deduce that ${}^{\lhd}[\![\mathbf{e}]\!] = {}^{\lhd}[\![\mathbf{f}]\!]$ thanks to the above characterisation. □

2.3 Confluence of the reduction relation

When considering $\rhd$ as a rewriting relation, we wondered whether it is confluent or not. It turns out that it is confluent. This property is not required for the proofs to come, but we include the result here for the sake of completeness.

First, notice that the irreflexive part of this relation is terminating: either $|uw\overline{w}wv| > |uwv|$, or $w = \varepsilon$ and the two words are equal. By Newman's lemma [5], it thus suffices to establish local confluence:

Lemma 2.12 (Local confluence)**.** *Let $m, m_1, m_2 \in \mathbf{X}^\star$ such that $m \rhd m_1$ and $m \rhd m_2$. There exists $n \in \mathbf{X}^\star$ such that $m_1 \rhd^\star n$ and $m_2 \rhd^\star n$.* ■

As we were trying to prove this lemma, it became obvious that the proof would be long and repetitive: the only strategy we could find was an exhaustive study of all critical pairs, and there are many of them. We thus used the proof assistant Coq to help us in that task:

- the statement of the lemma is short and involves very few notions, so that the encoding of the problem in Coq was immediate;
- this allowed us to get assurance that no critical pair was overlooked, something difficult to achieve with a pen-and-paper case analysis: there are thirty-five key cases, and this disjunction is not trivial to establish;
- we were able to mechanise a significant part of the proof, using the tactic language of Coq to implement pattern recognition and automatic reduction. For instance, in a case like
$$\begin{cases} m = u_1 w_1 \overline{w_1} w_1 v_1 \\ w_1 = x w_2 \overline{w_2} w_2 y \\ m_1 = u_1 x w_2 \overline{w_2} w_2 y v_1 \\ m_2 = u_1 w_1 \overline{w_1} x w_2 y u_1, \end{cases}$$
one only needs to recognise the patterns in m_1 and m_2 and reduce them as much as possible using relation $\rhd$ to get $n = u_1 x w_2 y v_1$.

Then the proof is just an exploration of the possible sub-cases, with the automatic tactic dealing swiftly with the administrative cases, and leaving only the subtle cases to be explicitly proven. Of the more than thirty-five cases that arise naturally, seventeen are ruled-out using size considerations, and twelve are mere questions of rewriting that were solved automatically. All in all, we only had to solve six sub-cases by hand, by using appropriate pumping lemmas.

By instrumenting the proof script to keep track of the uses of the relation $\rhd$, we were actually able to establish a slightly stronger result: four steps suffice to join all critical pairs.

Lemma 2.13. *Let $m, m_1, m_2 \in \mathbf{X}^\star$ such that $m \rhd m_1$ and $m \rhd m_2$. There exists $n \in \mathbf{X}^*$ and $k_1, k_2 \leqslant 4$ such that $m_1 \rhd^{k_1} n$ and $m_2 \rhd^{k_2} n$.* ■

Corollary 2.14 (Confluence)**.** *Let $m, m_1, m_2 \in \mathbf{X}^\star$ such that $m \rhd^\star m_1$ and $m \rhd^\star m_2$. There exists $n \in \mathbf{X}^*$ such that $m_1 \rhd^\star n$ and $m_2 \rhd^\star n$.* ■

The Coq proof is available online [12]; note that the converse of the reduction relation $\rhd$ is also confluent, albeit not terminating; this is much easier to prove.

2.4 Closure of an automaton

The problem here is the following: given two regular expressions $e, f \in \mathrm{Reg}^{\smile}\langle X \rangle$, how to decide ${}^{\lhd}[\![\mathbf{e}]\!] = {}^{\lhd}[\![\mathbf{f}]\!]$? We follow the approach proposed by Bloom et al.: given an automaton recognising a language L, we show how to construct an automaton recognising ${}^{\lhd}L$. To solve the initial problem, it then suffices to build two automata recognising $[\![\mathbf{e}]\!]$ and $[\![\mathbf{f}]\!]$, to apply a construction to obtain two automata for ${}^{\lhd}[\![\mathbf{e}]\!]$ and ${}^{\lhd}[\![\mathbf{f}]\!]$, and to check those for language equivalence.

As a starting point, we first recall the construction proposed in [6].

2.4.1 Original construction

This construction uses the transition monoid of the input automaton, which is assumed to be deterministic. It is a simple observation that in this case the relation induced by a word u on the states of the automaton (written $\widehat{\Delta}(u)$ in Definition 1.20) is in fact a function. Following [6], we denote this function by $u_{\mathscr{A}}$ in the following.

This monoid is finite, and its subsets form a Kleene Algebra. Bloom et al. [6] then proceed to define the closure automaton in the following way:

Theorem 2.15 (Closure automaton of [6])**.** *Let $L \subseteq \mathbf{X}^\star$ be a regular language, recognised by the deterministic automaton $\mathscr{A} = \langle Q, \mathbf{X}, q_0, Q_f, \delta\rangle$. Let $M_{\mathscr{A}}$ be the transition monoid of $\mathscr{A}$. Then the following deterministic automaton recognises ${}^{\triangleleft}L$:*

$$\begin{aligned} \mathscr{B} &:= \langle \mathcal{P}(M_{\mathscr{A}}) \times \mathcal{P}(M_{\mathscr{A}}), \mathbf{X}, \langle \{\varepsilon_{\mathscr{A}}\}, \{\varepsilon_{\mathscr{A}}\}\rangle, \mathscr{T}, \delta_1\rangle \\ \text{with } \mathscr{T} &:= \{\langle F, G\rangle \mid \exists u_{\mathscr{A}} \in F : u_{\mathscr{A}}(q_0) \in Q_f\} \quad , \\ \text{and } \delta_1(\langle F, G\rangle, x) &:= \left\langle F \cdot \{x_{\mathscr{A}}\} \cdot \left(\left(\{x'_{\mathscr{A}}\} \cdot G \cdot \{x_{\mathscr{A}}\}\right)^\star\right),\ \left(\{x'_{\mathscr{A}}\} \cdot G \cdot \{x_{\mathscr{A}}\}\right)^\star \right\rangle \quad . \end{aligned}$$ ■

An important idea in this construction, which leads to the presented one, is the transition rule for the second component above. Let us write $\delta_2(G, x)$ for the expression $(\{x'_{\mathscr{A}}\} \cdot G \cdot \{x_{\mathscr{A}}\})^\star$, so that the definition of δ_1 can be reformulated as

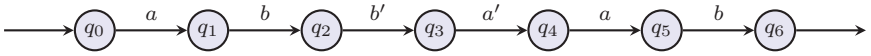
$$\delta_1(\langle F, G\rangle, x) = \langle F \cdot \{x_{\mathscr{A}}\} \cdot \delta_2(G, x),\ \ \delta_2(G, x)\rangle .$$

With that in mind, one can see the second component as some kind of *history*, that runs on its own, and is used at each step to enrich the first component. At this point, it might be interesting to notice that the formula for $\delta_2(G, x)$ closely resembles the definition of $\Gamma(wx)$ we gave in Section 2.2.2: $\Gamma(wx) = (x'\Gamma(w)x)^\star$.

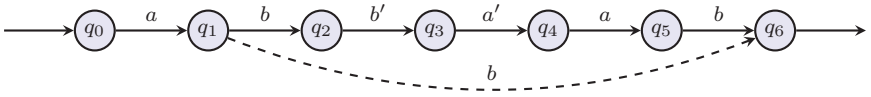
2.4.2 Intuitions

Let us try to build a closure automaton. One way would be to simply add transitions to the initial automaton. This idea comes naturally when one realises that if $u \rhd^\star v$, then v is obtained by erasing some subwords from u: at each reduction step $u_1 w\overline{w}wu_2 \rhd u_1 wu_2$ we just erase $\overline{w}w$. To "erase" such subwords using an automaton, it suffices to allow one to jump along certain paths.

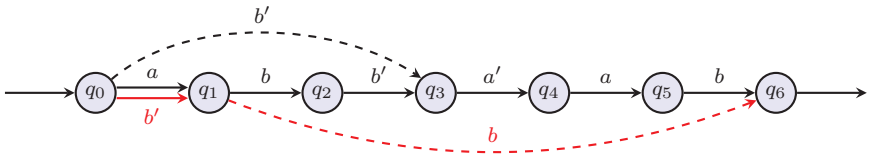
Suppose for instance that we start from the following automaton:



We can detect the pattern $ab\overline{ab}ab$, and allow one to "jump" over it when reading the last letter of the root of the pattern, in this case the b in second position. The automaton thus becomes:



However, this approach is too naive, and it quickly leads to errors. If for instance we slightly modify the above example by adding a transition labelled by b' between q_0 and q_1, the same method leads to the following automaton, by detecting the patterns $b'bb'$ between q_0 and q_3 and $abb'a'ab$ between q_0 and q_6.



The problem is that the word $b'b$ is now wrongly recognised in the produced automaton. What happens here is that we can use the jump from q_1 to q_6, even though we did not read the prerequisite for doing so, in this case the a constituting the beginning of the root ab of pattern $ab\overline{ab}ab$. (Note that the dual idea, consisting in enabling a jump when reading the first letter of the root of the pattern, would lead to similar problems.)

A way to prevent that, which was implicitly introduced in the original construction, consists in using a notion of *history*. The states of the closure automaton will be pairs of a state in the initial automaton and a history. That will allow us to distinguish between the state q_1 after reading a and the state q_1 after reading b', and to specify which jumps are possible considering what has been read previously. In the construction given in [6], the history is given by an element of $\mathcal{P}(M_{\mathscr{A}})$, in the second component of the states (the "G" part). We will define a history as a relation over states containing the jumps we are allowed to make after having read some word w, using $\Gamma(w)$.

2.4.3 New construction

We have shown in Proposition 2.10 that $u \in \Gamma(w) \Leftrightarrow \exists v \in suffixes(w) : u \rhd^\star \overline{v}v$, so we do have a characterisation of the words "we are allowed to jump over" after having read some word w. The problem is that we want a finite number of possible histories, and there are infinitely many $\Gamma(w)$ (for instance, all the $\Gamma(a^n)$ are different). To get that, we will project $\Gamma(w)$ on the automaton. Let us consider a non-deterministic automaton $\mathscr{A} = \langle Q, \mathbf{X}, I, \mathscr{T}, \Delta\rangle$ recognising a language L.

Definition 2.16 (History of a word)**.**
Let $\mathcal{R}^\star(Q)$ be the set of reflexive transitive binary relations over states of $\mathscr{A}$. Given a letter $x \in \mathbf{X}$, we denote by h_x the following increasing function on binary relations over states:

$$\begin{array}{rccc} h_x: & \mathcal{R}^\star(Q) & \to & \mathcal{R}^\star(Q) \\ & R & \mapsto & (\Delta(x') \cdot R \cdot \Delta(x))^\star \end{array}$$

For any word $w \in \mathbf{X}^\star$ we call *history* of the word w the relation $[w]$ defined inductively by

$$[\varepsilon] := \mathrm{Id}_Q \qquad\qquad [wx] := h_x([w]). \qquad *$$

One can notice right away the strong relationship between $[\cdot]$ and Γ:

Proposition 2.17. $\forall w, q_1, q_2,\ \langle q_1, q_2\rangle \in [w] \ \Leftrightarrow\ \exists u \in \Gamma(w) : q_1 \xrightarrow{u}_{\mathscr{A}} q_2.$ ■

This result is straightforward once one realises that $[w] = \widehat{\Delta}(\Gamma(w))$. By composing Propositions 2.10 and 2.17 we eventually obtain that $\langle q_1, q_2\rangle \in [w]$ if and only if $\exists u : q_1 \xrightarrow{u}_{\mathscr{A}} q_2$ and $u \rhd^\star \overline{v}v$, with v a suffix of w.

We now have all the tools required to define an automaton for the closure of $\mathscr{A}$:

Theorem 2.18 (Closure Automaton)**.** *The closure of the language L is recognised by the automaton*

$$\begin{array}{rl} & \mathscr{A}' := \langle Q \times \mathcal{R}^\star(Q), \mathbf{X}, I \times \{Id_Q\}, \mathscr{T} \times \mathcal{R}^\star(Q), \Delta'\rangle \ , \\ \textit{where} & \Delta' := \{\langle\langle q_1, R\rangle, x, \langle q_2, h_x(R)\rangle\rangle \mid \langle q_1, q_2\rangle \in \Delta(x) \cdot h_x(R)\} \ . \end{array}$$ ■

We shall write L' for the language recognised by $\mathscr{A}'$. One can read the set of transitions as "from a state q_1 with a history R, perform a step x in the automaton $\mathscr{A}$, and then a jump compatible with $h_x(R)$, which becomes the new history". An example of this construction is given in Section 2.4.4. It is useful to notice at this point that if $\langle p, R\rangle$ is an accessible state in $\mathscr{A}'$, then there is some word u such that $\langle i, \mathrm{Id}_Q\rangle = \langle i, [\varepsilon]\rangle \xrightarrow{u}_{\mathscr{A}'} \langle p, R\rangle$, from which we can deduce by induction on u that $R = [u]$. One can see, from the definition of Δ' and Proposition 2.17 that:

$$\exists\, \langle q_2, v\rangle \in Q \times \Gamma(ux) : q_1 \xrightarrow{x}_{\mathscr{A}} q_2 \xrightarrow{v}_{\mathscr{A}} q_3 \;\Leftrightarrow\; \langle q_1, [u]\rangle \xrightarrow{x}_{\mathscr{A}'} \langle q_3, [ux]\rangle\,. \tag{2.15}$$

Now we prove the correctness of this construction. The following property of $[\cdot]$ is proved by exhibiting a simulation:

Proposition 2.19. *For all words $u, v \in \mathbf{X}^\star$ such that $u \rhd v$, we have $[u] \subseteq [v]$.* ■

Proof. First, notice that $\Gamma(u) \subseteq \Gamma(v) \Rightarrow [u] \subseteq [v]$, using Proposition 2.17. Indeed, if $\Gamma(u) \subseteq \Gamma(v)$, then for any states $p, q \in Q$ we have

$$\begin{aligned}
\langle p, q\rangle \in [u] &\Leftrightarrow \exists w \in \Gamma(u) : p \xrightarrow{w}_{\mathscr{A}} q && \text{(Proposition 2.17)}\\
&\Rightarrow \exists w \in \Gamma(v) : p \xrightarrow{w}_{\mathscr{A}} q && (\Gamma(u) \subseteq \Gamma(v))\\
&\Leftrightarrow \langle p, q\rangle \in [v]. && \text{(Proposition 2.17)}
\end{aligned}$$

It thus suffices to prove $u \rhd v \Rightarrow \Gamma(u) \subseteq \Gamma(v)$, which can be rewritten as $\Gamma(u_1 w \overline{w} w u_2) \subseteq \Gamma(u_1 w u_2)$. We can drop u_2 (it is clear that $\Gamma(w_1) \subseteq \Gamma(w_2) \Rightarrow \forall x \in \mathbf{X}, \Gamma(w_1 x) \subseteq \Gamma(w_2 x)$, from the definition of Γ): we now have to prove that $\Gamma(u_1 w \overline{w} w) \subseteq \Gamma(u_1 w)$. The proof of this inclusion relies on the fact that the automaton $\mathscr{G}(u_1 w \overline{w} w)$ is simulated by the automaton $\mathscr{G}(u_1 w)$.

First, we give in Figure 2.4 an abstract view of the automata recognising $\Gamma(uw\overline{w}w)$ and $\Gamma(uw)$ defined as before. With the notations of this figure, now define a relation $\preccurlyeq$ as follows (this relation is also represented in dashed lines in Figure 2.4):

$$\begin{aligned}
a_i &\preccurlyeq b_i && \text{for all } i \leqslant n+m\ ,\\
a_{n+m+i} &\preccurlyeq b_{n+m-i} && \text{for all } i \leqslant n\ ,\\
a_{2n+m+i} &\preccurlyeq b_{m+i} && \text{for all } i \leqslant n\ ;
\end{aligned}$$

One easily checks that this relation is a simulation, thus establishing in particular that the language recognised by the left-hand side automaton (for $\Gamma(uw\overline{w}w)$) is contained in that of the right-hand side (for $\Gamma(uw)$). □

We define an order relation $\preccurlyeq$ on the states of the produced automaton $(Q \times \mathcal{R}^\star(Q))$, by $\langle p, R\rangle \preccurlyeq \langle q, S\rangle$ when $p = q$ and $R \subseteq S$.

Proposition 2.20. *The relation $\preccurlyeq$ is a simulation for the automaton $\mathscr{A}'$.* ■

Proof. Suppose that $\langle p, R\rangle \preccurlyeq \langle q, S\rangle$ and $\langle p, R\rangle \xrightarrow{x}_{\mathscr{A}'} \langle p', h_x(R)\rangle$, i.e., p and p' are related by $\Delta(x) \cdot h_x(R)$. We have $p = q$ and $R \subseteq S$, hence $h_x(R) \subseteq h_x(S)$, and thus p and p' are also related by $\Delta(x) \cdot h_x(S)$ meaning that $\langle p, S\rangle \xrightarrow{x}_{\mathscr{A}'} \langle p', h_x(S)\rangle$. It remains to check that $\langle p', h_x(R)\rangle \preccurlyeq \langle p', h_x(S)\rangle$, i.e., $h_x(R) \subseteq h_x(S)$, which we just proved. □

We may now prove that $L' = {}^{\triangleleft}L$.

Lemma 2.21. $L' \subseteq {}^{\triangleleft}L$ ■

Proof. We prove by induction on u that for all q_0, q such that $\langle q_0, [\varepsilon]\rangle \xrightarrow{u}_{\mathscr{A}'} \langle q, [u]\rangle$, there exists v such that $v \rhd^\star u$ and $q_0 \xrightarrow{v}_{\mathscr{A}} q$. The case $u = \varepsilon$ is trivial.

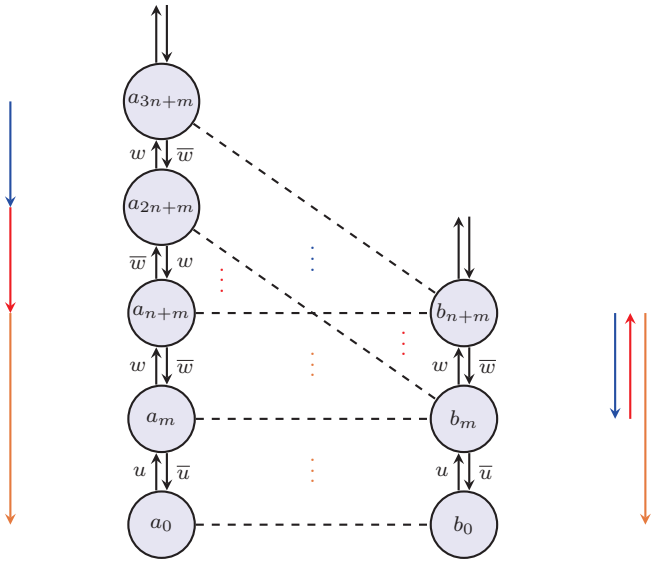


Figure 2.4: Automata $\mathscr{G}(uw\overline{w}w)$ and $\mathscr{G}(uw)$, with $|u| = m$ and $|w| = n$

If $\langle q_0, [\varepsilon]\rangle \xrightarrow{u}_{\mathscr{A}'} \langle q_1, [u]\rangle \xrightarrow{x}_{\mathscr{A}'} \langle q, [ux]\rangle$, by induction there exists a word v_1 such that $q_0 \xrightarrow{v_1}_{\mathscr{A}} q_1$ and $v_1 \rhd^\star u$. We also know (by (2.15) and Proposition 2.10) that there are some q_2, v_2 and $v_3 \in suffixes(ux)$ such that $q_1 \xrightarrow{x}_{\mathscr{A}} q_2$, $v_2 \rhd^\star \overline{v}_3 v_3$ and $q_2 \xrightarrow{v_2}_{\mathscr{A}} q$. We get:

$$q_0 \xrightarrow{v_1}_{\mathscr{A}} q_1 \xrightarrow{x}_{\mathscr{A}} q_2 \xrightarrow{v_2}_{\mathscr{A}} q \text{ and } v_1 x v_2 \rhd^\star ux v_2 \rhd^\star ux\overline{v}_3 v_3 \rhd ux.$$

By choosing $q \in \mathscr{T}$, we obtain the desired result. □

Lemma 2.22. *$L \subseteq L'$* ■

Proof. First notice that for all $R \in \mathcal{R}^\star(Q)$, $h_x(R)$ is a reflexive relation, hence $q_1 \xrightarrow{x}_{\mathscr{A}} q_2$ entails $\forall R, \langle q_1, R\rangle \xrightarrow{x}_{\mathscr{A}'} \langle q_2, h_x(R)\rangle$. This means that the binary relation S defined by $p\ S\ \langle q, R\rangle \Leftrightarrow p = q$ is a simulation between $\mathscr{A}$ and $\mathscr{A}'$, and thus

$$L = L(\mathscr{A}) \subseteq L(\mathscr{A}') = L'.$$ □

Lemma 2.23. *L' is downward-closed for $\rhd$.* ■

A technical lemma is required to establish this closure property:

Lemma 2.24. *If $\langle q_1, [uw]\rangle \xrightarrow{x}_{\mathscr{A}'} \langle q_2, [uwx]\rangle \xrightarrow{\overline{wx}\ wx}_{\mathscr{A}'} \langle q_3, [uwx\ \overline{wx}\ wx]\rangle$, then we have $\langle q_1, [uw]\rangle \xrightarrow{x}_{\mathscr{A}'} \langle q_3, [uwx]\rangle$.* ■

Proof. If $|w| = n$ and $|u| = m$, the premise can be equivalently stated:

$$\langle q_1, [(uw)|_{m+n-1}]\rangle \xrightarrow{w(n)}_{\mathscr{A}'} \langle q_2, [uw]\rangle \xrightarrow{\overline{w}w}_{\mathscr{A}'} \langle q_3, [uw\overline{w}w]\rangle .$$

(Recall that $u|_i$ denotes the prefix of length i of a word u.) Let us write $\Gamma_i = \Gamma\left(uw(\overline{w}w)|_i\right)$ and $x_i = (uw\overline{w}w)(n+m+i)$ for $0 \leqslant i \leqslant 2n$. Notice that $\Gamma_i = \Gamma\left((uw\overline{w}w)|_{m+n+i}\right)$. By Proposition 2.17 and the definition of $\mathscr{A}'$, we can show that there are $v_i \in \Gamma_i$ such that the execution above can be lifted into an execution in $\mathscr{A}$:

$$q_1 \xrightarrow{x_0 v_0 x_1 v_1 \cdots x_i v_i \cdots x_{2n} v_{2n}}_{\mathscr{A}} q_3.$$

Then one can prove using Proposition 2.10 that:

$$\forall i, \exists t_i \in \Gamma(uw) : (\overline{w}w)|_i v_i \rhd^\star t_i (\overline{w}w)|_i. \tag{2.16}$$

As v_i is in $\Gamma\left(uw(\overline{w}w)|_i\right)$, we know that there is some suffix t of $uw(\overline{w}w)|_i$ such that $v_i \rhd^\star \overline{t}t$. We will do a case analysis on the size of t:

- if $n+i \leqslant |t|$, then there is a suffix s of u such that $t = sw(\overline{w}w)|_i$, so

$$(\overline{w}w)|_i v_i \rhd^\star (\overline{w}w)|_i \overline{(\overline{w}w)|_i}\,\overline{w}\ \bar{s}sw(\overline{w}w)|_i.$$

  - If $i < n$ then there is a word p such that $\overline{w} = (\overline{w}w)|_i p$ so

$$\begin{aligned}(\overline{w}w)|_i v_i &\rhd^\star (\overline{w}w)|_i \overline{(\overline{w}w)|_i}(\overline{w}w)|_i p\bar{s}sw(\overline{w}w)|_i \\ &\rhd (\overline{w}w)|_i p\bar{s}sw(\overline{w}w)|_i = \overline{s\overline{w}}sw(\overline{w}w)|_i.\end{aligned}$$

  - Otherwise we can write $(\overline{w}w)|_i = \overline{w}w_1$ and $w = w_1w_2$, so

$$\begin{aligned}(\overline{w}w)|_i v_i &\rhd^\star \overline{w}w_1\overline{\overline{w}w_1}\,\overline{w}\ \bar{s}sw(\overline{w}w)|_i = \overline{w}_2\ \overline{w}_1w_1\overline{w}_1w\overline{w}\ \bar{s}sw(\overline{w}w)|_i \\ &\rhd \overline{w}_2\ \overline{w}_1w\overline{w}\ \bar{s}sw(\overline{w}w)|_i = \overline{w}w\overline{w}\ \bar{s}sw(\overline{w}w)|_i \\ &\rhd \overline{s\overline{w}}sw(\overline{w}w)|_i.\end{aligned}$$

  As $s \in suffixes(u)$ we know that $sw \in suffixes(uw)$, hence $\overline{s\overline{w}}sw \in \Gamma(uw)$.

- If $i \leqslant |t| < n+i$ then $w = w_1w_2$ and $t = w_2(\overline{w}w)|_i$ so

$$(\overline{w}w)|_i v_i \rhd^\star (\overline{w}w)|_i \overline{(\overline{w}w)|_i}\,\overline{w_2}w_2(\overline{w}w)|_i$$

  - If $i < n$ then there is a word p such that $\overline{w} = (\overline{w}w)|_i p$. As $\overline{w} = \overline{w_2}\ \overline{w_1}$, we can also compare $(\overline{w}w)|_i$ with $\overline{w_2}$:
    - If $(\overline{w}w)|_i = \overline{w_2}w_3$ then

$$\begin{aligned}(\overline{w}w)|_i v_i &\rhd^\star \overline{w_2}w_3\overline{w_3}w_2\overline{w_2}w_2(\overline{w}w)|_i \\ &\rhd \overline{w_2}w_3\overline{w_3}w_2(\overline{w}w)|_i = (\overline{w}w)|_i\overline{(\overline{w}w)|_i}(\overline{w}w)|_i \\ &= \overline{\overline{(\overline{w}w)|_i}}\ \overline{(\overline{w}w)|_i}(\overline{w}w)|_i\end{aligned}$$

      And as $\overline{w} = (\overline{w}w)|_i p$, $w = \overline{p}\overline{(\overline{w}w)|_i}$ we know that

$$\overline{(\overline{w}w)|_i} \in suffixes(w) \subseteq suffixes(uw),$$

      hence $\overline{\overline{(\overline{w}w)|_i}}\ \overline{(\overline{w}w)|_i} \in \Gamma(uw)$.
    - If on the other hand $\overline{w_2} = (\overline{w}w)|_i w_3$, we have

$$\begin{aligned}(\overline{w}w)|_i v_i &\rhd^\star (\overline{w}w)|_i\overline{(\overline{w}w)|_i}(\overline{w}w)|_i w_3\overline{w_3}\overline{(\overline{w}w)|_i}(\overline{w}w)|_i \\ &\rhd (\overline{w}w)|_i w_3\overline{w_3}\overline{(\overline{w}w)|_i}(\overline{w}w)|_i = \overline{w_2}w_2(\overline{w}w)|_i\end{aligned}$$

      $w_2 \in suffixes(w) \subseteq suffixes(uw)$ so $\overline{w}_2w_2 \in \Gamma(uw)$.
  - Otherwise we can write $(\overline{w}w)|_i = \overline{w}w_3$ and $w = w_3w_4$, so

$$\begin{aligned}(\overline{w}w)|_i v_i &\rhd^\star \overline{w}w_3\overline{w_3}w_3w_4\overline{w_2}w_2(\overline{w}w)|_i \\ &\rhd \overline{w}w_3w_4\overline{w_2}w_2(\overline{w}w)|_i = \overline{w}w_1w_2\overline{w_2}w_2(\overline{w}w)|_i \\ &\rhd \overline{w}w_1w_2(\overline{w}w)|_i = \overline{w}w(\overline{w}w)|_i\end{aligned}$$

    And obviously $\overline{w}w \in \Gamma(uw)$.

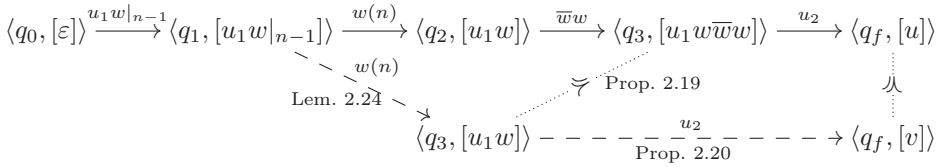
- If $|t| < i$ then $(\overline{w}w)|_i = st$. In this case we have $(\overline{w}w)|_i v_i \rhd^\star st\bar{t}t \rhd st = \bar{\varepsilon}\varepsilon(\overline{w}w)|_i$, and $\varepsilon \in suffixes(uw)$ so $\bar{\varepsilon}\varepsilon \in \Gamma(uw)$.

In all cases, we have shown that $(\overline{w}w)_i v_i \rhd^\star t_i(\overline{w}w)|_i$ with $t_i \in \Gamma(uw)$.

We deduce that $v_0x_1v_1\cdots x_iv_i\cdots x_{2n}v_{2n} \rhd^\star t_0t_1\cdots t_{2n}\overline{w}w \in \Gamma(uw)^{2n+2} \subseteq \Gamma(uw)$. By Proposition 2.10, this means that $v_0x_1v_1\cdots x_iv_i\cdots x_{2n}v_{2n}$ is in $\Gamma(uw)$, so that

$$\langle q_1, q_3\rangle \in \Delta(w(n))\cdot[uw],\ \textit{and}\ \langle q_1, [uw|_{n-1}]\rangle \xrightarrow{w(n)}_{\mathscr{A}'} \langle q_2, [uw]\rangle\,. \qquad \square$$

With this intermediate lemma, one can obtain a succinct proof of Lemma 2.23:
Proof. The statement of the lemma is equivalent to saying that if $u \rhd v$ with $u \in L'$ then v is also in L'. Consider $u = u_1 w \cdot \overline{w} \cdot w u_2$ and $v = u_1 w u_2$ with $|w| = n \geqslant 1$ (the case where $w = \varepsilon$ does not hold any interest since it implies that $u = v$). By combining Lemma 2.24, Proposition 2.19 and Proposition 2.20 we can build the following diagram:



□

Lemmas 2.22 and 2.23 tell us that L' is closed and contains L, so by definition of the closure of a language, we get ${}^{\lhd}L \subseteq L'$. Lemma 2.21 gives us the other inclusion, thus proving Theorem 2.18.

2.4.4 Example

Let us illustrate this construction on a simple example: consider the automaton depicted in Figure 2.5. It recognises the language $[\![(a + b')b(b'a')^\star ab]\!]$, which is not closed. Informally, we observe that

- when starting with an a, and by firing the starred expression only once, a pattern of root ab appears, so that the word ab should belong to the closure. Such a behaviour is no longer possible if we fire the starred expression more than once;
- when starting with a b, and by firing the starred expression at least once, a pattern of root b' appears, so that the language $[\![b'a'(b'a')^\star ab]\!]$ is contained in the closure.

Now the first step to build the closure automaton consists in computing the values of $[\cdot]$; they are summarised in Figure 2.6. We can then build the closure automaton as described in Theorem 2.18. The resulting non-deterministic automaton is drawn in Figure 2.7. Due to the history component, the states B and C are duplicated; moreover, "jumps" have been used to obtain the two red transitions, from $\langle A, [\varepsilon]\rangle$ to $\langle D, [b']\rangle$, and from $\langle B, [a]\rangle$ to $\langle F, [ab]\rangle$. Using those transitions, one can notice that the word ab is now accepted, as well as all words from $[\![b'a'(b'a')^\star ab]\!]$.

The determinised version of this automaton is finally given in Figure 2.8.

2.5 Analysis and consequences

2.5.1 Relationship with Bloom et al's construction

As suggested by an anonymous referee, one can also formally relate our construction to the one from [6]: we give below an explicit and rather natural bisimulation relation between the automata produced by both these methods. This results in an alternative correctness proof of the proposed construction, by reducing it to the correctness of the one from [6].

We first make the two constructions comparable: the original construction, because it considers the transition monoid, takes as input a deterministic automaton. It returns a deterministic automaton. Instead, our construction does not require determinism in its input, but produces a non-deterministic automaton. We thus have to ask of both methods to accept as their input a *non-deterministic* automaton, and to return a *deterministic* automaton.

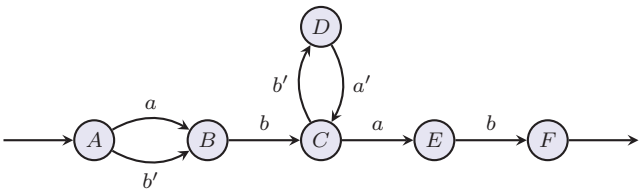


Figure 2.5: Initial automaton.

$$
\begin{array}{ll}
[\,\varepsilon\,] = \mathrm{Id}_Q & (= [a'] = [aa'] = [ba'] = [aba'])\\
[\,a\,] = \mathrm{Id}_Q \cup \{\langle D, E\rangle\} & (= [aa] = [b'a] = [ba] = [aba])\\
[\,b\,] = \mathrm{Id}_Q \cup \{\langle A, C\rangle\} & (= [bb] = [b'b] = [b'a'] = [abb])\\
[\,b'\,] = \mathrm{Id}_Q \cup \{\langle B, D\rangle\} & (= [ab'] = [bb'] = [b'b'] = [abb'])\\
[ab] = \mathrm{Id}_Q \cup \{\langle A, C\rangle, \langle C, F\rangle, \langle A, F\rangle\}
\end{array}
$$

Figure 2.6: Computation of $[\cdot]$.

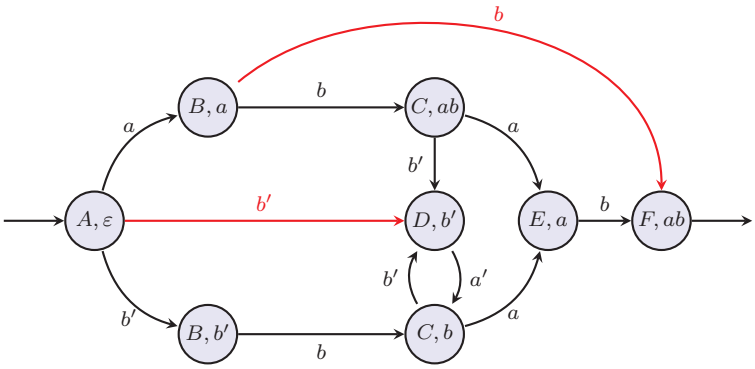


Figure 2.7: Closure automaton.

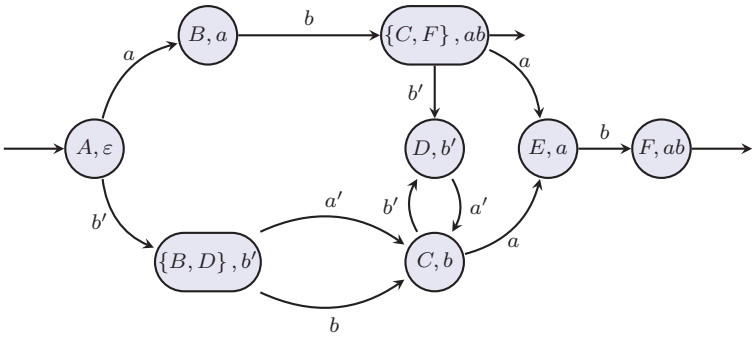


Figure 2.8: Determinised version.

For our construction, the straightforward thing to do would be to determinise the automaton afterwards. We can actually do better, by noticing that from a state $\langle p, [u]\rangle$, reading some letter x, there may be a lot of accessible states, but all of their histories (second components) will be equal to $[ux]$. So in order to get a deterministic automaton, one only has to perform the power-set construction on the first component of the automaton. This way, we get an automaton $\mathscr{A}_1$ with states in $\mathcal{P}(Q) \times \mathcal{R}^\star(Q)$ and a transition function

$$\delta_1(\langle P, R\rangle, x) = \langle P \cdot (\Delta(x) \cdot h_x(R)), h_x(R)\rangle .$$

The original construction can also be adjusted quite easily: first build a deterministic automaton $\mathscr{D}$ with the usual powerset construction, then apply the construction as described in Theorem 2.15 to get an automaton which we call $\mathscr{A}_2$. An important thing here is to understand the shape of the resulting transition monoid $M_{\mathscr{D}}$: its elements are functions over sets of states (because of the power-set construction) induced by words; more precisely, they are sup-semilattice homomorphisms, and they are in bijection with binary relations on states induced by words. (The relation induced by u is the set of pairs $\langle p, q\rangle$ such that $p \xrightarrow{u} q$.)

Define the following KA-homomorphism from $\mathcal{P}(M_{\mathscr{D}})$ to $\mathcal{P}(Q^2)$:

$$i(F) = \{\langle p, q\rangle \mid \exists u_{\mathscr{D}} \in F : q \in u_{\mathscr{D}}(\{p\})\} \quad .$$

(That i is a KA-homomorphism comes from the fact that the elements of $M_{\mathscr{D}}$ are themselves sup-semilattice homomorphisms on $\mathcal{P}(Q)$.) We can check that for all $x \in \mathbf{X}$, we have

$$\begin{aligned} i(\{x_{\mathscr{D}}\}) &= \{\langle p, q\rangle \mid q \in x_{\mathscr{D}}(\{p\})\} = \{\langle p, q\rangle \mid q \in \delta(\{p\}, x)\} \\ &= \left\{\langle p, q\rangle \;\middle|\; p \xrightarrow{x}_{\mathscr{A}} q\right\} = \Delta(x) \ , \end{aligned}$$

The following lemma follows:

Lemma 2.25. *The relation below is a bisimulation between $\mathscr{A}_1$ and $\mathscr{A}_2$.*

$$\{\langle\langle I \cdot i(F), i(G)\rangle, \ \langle F, G\rangle\rangle \mid \forall F, G\}$$

■

Proof. Consider a non-deterministic automaton $\mathscr{A} = \langle Q, \mathbf{X}, I, Q_f, \Delta\rangle$. Its determinisation is $\mathscr{D} = \langle \mathcal{P}(Q), \mathbf{X}, I, \mathscr{T}, \delta\rangle$ with

$$\mathscr{T} = \{P \mid P \cap Q_f \neq \emptyset\} \text{ and } \delta(P, x) = P \cdot \Delta(x).$$

We can build two automata recognising its closure. The first one, derived from our construction, is $\mathscr{A}_1 = \langle \mathcal{P}(Q) \times \mathcal{R}^\star(Q), \mathbf{X}, \langle I, \mathrm{Id}_Q\rangle, \mathscr{T}_1, \delta_1\rangle$ where:

- $\mathcal{R}^\star(Q)$ is the set reflexive transitive relations over Q,
- $\mathscr{T}_1 := \{\langle P, R\rangle \mid P \cap Q_f \neq \emptyset, \ R \in \mathcal{R}^\star(Q)\}$,
- and $\delta_1(\langle P, R\rangle, x) := \langle P \cdot h_x(R), h_x(R)\rangle$.

The one given by the original construction is $\mathscr{A}_2 = \langle \mathcal{P}(M_{\mathscr{D}}) \times \mathcal{P}(M_{\mathscr{D}}), \mathbf{X}, \langle \underline{\varepsilon}, \underline{\varepsilon}\rangle, \mathscr{T}_2, \delta_2\rangle$ where

- $M_{\mathscr{D}}$ is the transition monoid of $\mathscr{D}$, a set of endomorphisms of $\mathcal{P}(Q)$ induced by words,
- $\underline{w} := \{w_{\mathscr{D}}\}$ is a singleton containing the interpretation of a word w in $M_{\mathscr{D}}$,
- $\mathscr{T}_2 := \{\langle F, G\rangle \mid \exists q_f \in Q_f, \exists f \in F : q_f \in f(I)\}$,

- and $\delta_2(\langle F, G\rangle, x) := \langle F \odot \underline{x} \odot (\underline{x}' \odot G \odot \underline{x})^\star, (\underline{x}' \odot G \odot \underline{x})^\star\rangle$.

($A \odot B := \{g \cdot f \mid f \in A \wedge g \in B\}$.) The fact that the elements of $M_{\mathscr{D}}$ are semilattice-homomorphisms can be easily checked, as $u_{\mathscr{D}}(P)$ is the only state of $\mathscr{D}$ (*i.e.* a set of states of $\mathscr{A}$) such that $P \xrightarrow{u}_{\mathscr{D}} u_{\mathscr{D}}(P)$. Then is is straightforward that:

$$\begin{aligned} u_{\mathscr{D}}(P_1 \cup P_2) &= \{q \in Q \mid \exists p \in P_1 \cup P_2 : p \xrightarrow{u}_{\mathscr{A}} q\} \\ &= \{q \in Q \mid \exists p \in P_1 : p \xrightarrow{u}_{\mathscr{A}} q\} \cup \{q \in Q \mid \exists p \in P_2 : p \xrightarrow{u}_{\mathscr{A}} q\} \\ &= u_{\mathscr{D}}(P_1) \cup u_{\mathscr{D}}(P_2). \end{aligned}$$

Now, to give the bisimulation we need the following morphism i from $\mathcal{P}(M_{\mathscr{D}})$ to $\mathcal{P}(Q^2)$ defined by

$$i(F) := \{\langle p, q\rangle \mid \exists f \in F : q \in f(\{p\})\}.$$

Note that i is a KA-homomorphism because the elements of the transition monoid of the determinised automaton are semilattice-homomorphisms from $\mathcal{P}(Q)$ to $\mathcal{P}(Q)$. Let's check that:

$$\begin{aligned} \varepsilon_{\mathscr{D}} &= \mathrm{Id}_{\mathcal{P}(Q)}, \text{ meaning that } i(\underline{\varepsilon}) = \mathrm{Id}_Q; \\ i(F_1 \cup F_2) &= \{\langle p, q\rangle \mid \exists f \in F_1 \cup F_2 : q \in f(\{p\})\} \\ &= \{\langle p, q\rangle \mid \exists f \in F_1 : q \in f(\{p\})\} \cup \{\langle p, q\rangle \mid \exists f \in F_2 : q \in f(\{p\})\} \\ &= i(F_1) \cup i(F_2); \\ i(F_1 \odot F_2) &= \{\langle p, q\rangle \mid \exists f \in F_1 \odot F_2 : q \in f(\{p\})\} \\ &= \{\langle p, q\rangle \mid \exists \langle f, g\rangle \in F_1 \times F_2 : q \in g \cdot f(\{p\})\} \\ &= \{\langle p, q\rangle \mid \exists f \in F_1 : \exists p' \in f(\{p\}) : \exists g \in F_2 : q \in g(\{p'\})\} \\ &\qquad\qquad (g \text{ is a semilattice homomorphism}) \\ &= \{\langle p, q\rangle \mid \exists p' : \langle p, p'\rangle \in i(F_1) \wedge \langle p', q\rangle \in i(F_2)\} \\ &= i(F_1) \cdot i(F_2) \end{aligned}$$

For the * operation, recall that

$$\forall F \in \mathcal{P}(M_{\mathscr{D}}), \exists n_1(F) \in \mathbb{N} : \forall n_1(F) \leqslant m, F^\star = (F \cup \underline{\varepsilon})^m;$$

and that

$$\forall R \in \mathcal{R}el\langle Q\rangle, \exists n_2(R) \in \mathbb{N} : \forall n_2(R) \leqslant m, R^\star = (R \cup \mathrm{Id}_Q)^m.$$

Then, if we write $m = \max(n_1(F), n_2(u_{\mathscr{D}}(F)))$,

$$\begin{aligned} i(F^\star) &= i((F \cup \underline{\varepsilon})^m) \\ &= (i(F) \cup i(\underline{\varepsilon}))^m \\ &= (i(F))^\star \end{aligned}$$

We can also check that, for any $x \in \mathbf{X}$:

$$\begin{aligned} i(\underline{x}) &= \{\langle p, q\rangle \mid q \in x_{\mathscr{D}}(\{p\})\} \\ &= \{\langle p, q\rangle \mid q \in \delta(\{p\}, x)\} \\ &= \{\langle p, q\rangle \mid p \xrightarrow{x}_{\mathscr{A}} q\} \\ &= \Delta(x). \end{aligned}$$

The bisimulation $\sim$ can thus be expressed:

$$\sim := \{\langle\langle I \cdot i(F), i(G)\rangle, \langle F, G\rangle\rangle\}$$

where $\langle F, G\rangle$ are states of $\mathscr{A}_2$. We now prove that it is indeed a bisimulation.

1. We need the inital states to be related. This is obvious as $\varepsilon_{\mathscr{D}} = \mathrm{Id}_{\mathcal{P}(Q)}$, meaning that $i(\underline{\varepsilon}) = \mathrm{Id}_Q$. Furthermore, $[\varepsilon] = \mathrm{Id}_Q$ and $I = I \cdot \mathrm{Id}_Q$. That means $\langle I, [\varepsilon]\rangle \sim \langle \underline{\varepsilon}, \underline{\varepsilon}\rangle$.

2. For the final states, it isn't much more complicated:

$$\begin{aligned}\langle F, G\rangle \in \mathscr{T}_2 &\Leftrightarrow \exists q_f \in Q_f : \exists f \in F : q_f \in f(I)\\ &\Leftrightarrow \exists q_f \in Q_f : q_f \in I \cdot i(F)\\ &\Leftrightarrow I \cdot i(F) \cap Q_f \neq \emptyset\\ &\Leftrightarrow \langle I \cdot i(F), i(G)\rangle \in \mathscr{T}_1.\end{aligned}$$

3. What remains to be shown is that this relation is stable under transitions from both sides. Suppose that $\langle Q, R\rangle \sim \langle F, G\rangle$, and consider $x \in \mathbf{X}$. After reading x we get in $\mathscr{A}_2$ the state $\langle F \odot \underline{x} \odot G', G'\rangle$, with $G' = (\underline{x'} \odot G \odot \underline{x})^\star$, and in $\mathscr{A}_1$ the state $\langle Q \cdot (\Delta(x) \cdot h_x(R)), h_x(R)\rangle$. We will prove that they are still related in two steps, first by looking at the second component, and then dealing with the first one.

   a) We know that $R = i(G)$, and that $i(\underline{x}) = \Delta(x)$.

$$\begin{aligned}h_x(R) &= (\Delta(x') \cdot R \cdot \Delta(x))^\star\\ &= (i(\underline{x'}) \cdot i(G) \cdot i(\underline{x}))^\star\\ &= i(G') \qquad (i \text{ is a morphism})\end{aligned}$$

   b) Now the first component comes quite easily:

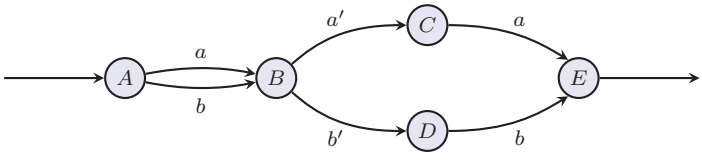
$$\begin{aligned}Q \cdot (\Delta(x) \cdot h_x(R)) &= (I \cdot i(F)) \cdot (i(\underline{x}) \cdot i(G'))\\ &= I \cdot (i(F) \cdot i(\underline{x}) \cdot i(G'))\\ &= I \cdot i(F \odot \underline{x} \odot G').\end{aligned}$$

□

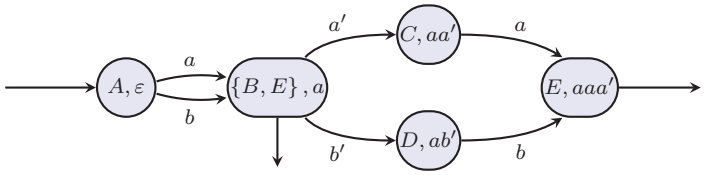
This tight relationship between both constructions may lead to believe the automata built by both constructions are isomorphic. However, it is not the case: the construction presented in this chapter produces a smaller automaton than the one given in [6]. Indeed, by unfolding the above bisimulation, one can find a surjective morphism from $\mathscr{A}_2$ to $\mathscr{A}_1$. But such a morphism cannot be found in general in the other direction, thus no bijection. This is illustrated in Example 2.26 below.

Intuitively, a major difference comes from the difference between the determinised automaton and the automaton induced by the right Cayley graph of the transition monoid. In a deterministic automaton, we can define an equivalence relation $u \sim v$, that holds if reading u and reading v in the automaton lead to the same state. In an automaton obtained by a power-set construction, $u \sim v$ is equivalent to saying that for any state p in the original automaton, p can be reached from the initial state by reading u if and only if it can be reached by reading v. In an automaton built by the monoid construction, $u \sim v$ corresponds to saying that for any pair of states p, q in the original automaton, there is a path from p to q labelled by u if and only if there is a path from p to q labelled by v. This second equivalence relation strictly contains the first one.

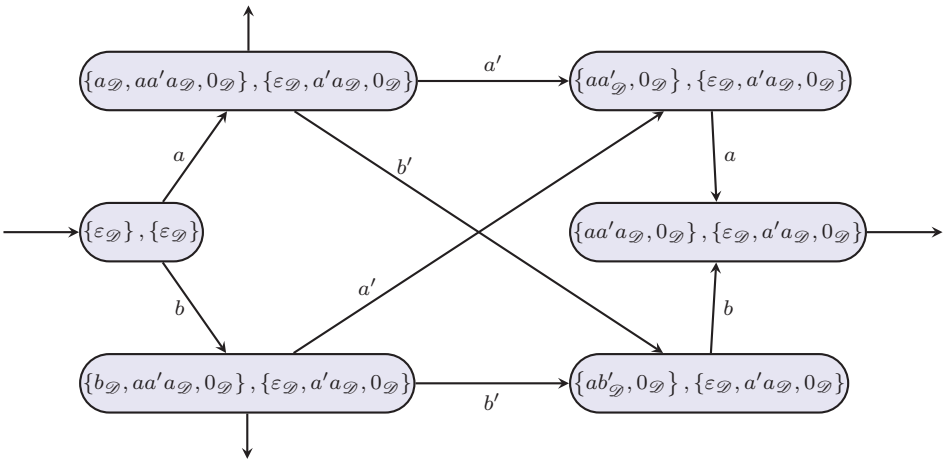
Example 2.26.
Consider the following deterministic automaton $\mathscr{D}$ over the alphabet $\{a, b, a', b'\}$:



By applying the determinised version of our construction, we build the following automaton:



However, the use of the method from [6] give rise to this automaton:

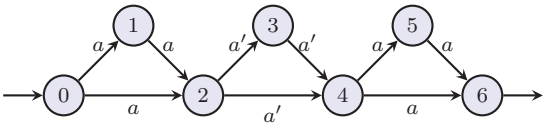


Notice that this automaton is just like the previous one, except that state $\langle \{B, E\}, [a] \rangle$ has been duplicated. This happens because:

$$a_{\mathscr{D}} = \Delta(a) = \{\langle A, B\rangle, \langle C, E\rangle\} \neq \{\langle A, B\rangle, \langle D, E\rangle\} = \Delta(b) = b_{\mathscr{D}}.$$

Thus the monoid will differentiate the state we get after reading a and the state we get after reading b. •

Another important difference is the fact that the construction in [6] uses *sets* of relations when we simply use relations. This is particularly visible when looking at the histories. Consider the following automaton $\mathscr{A}$:



The history induced by a in this automaton is

$$[a] = (\Delta(a') \cdot \Delta(a))^\star = \mathrm{Id}_Q \cup \{\langle 2, 6\rangle; \langle 2, 5\rangle; \langle 3, 5\rangle\};$$

and the one induced by aa is

$$[aa] = (\Delta(a') \cdot [a] \cdot \Delta(a))^\star = \mathrm{Id}_Q \cup \{\langle 2, 6\rangle; \langle 2, 5\rangle; \langle 3, 5\rangle\} = [a].$$

However, if we do this with the monoid approach, we can see that

$$a'a_{\mathscr{A}} = \{\langle 2, 6\rangle; \langle 2, 5\rangle; \langle 3, 5\rangle\} \neq \{\langle 2, 6\rangle\} = a'a'aa_{\mathscr{A}}.$$

Thus the history we get after a is $\{\varepsilon_{\mathscr{A}}; a'a_{\mathscr{A}}; 0_{\mathscr{A}}\}$ which is a different set that the one we get after aa which is $\{\varepsilon_{\mathscr{A}}; a'a_{\mathscr{A}}; a'a'aa_{\mathscr{A}}; 0_{\mathscr{A}}\}$. This will induce some duplication in the closure automaton. Indeed on this example, our deterministic closure automaton will have 7 states, whereas [6]'s construction produces an automaton with 11 states. In general, our construction always yields an automaton at least as small as the previous construction.

2.5.2 Complexity

A relevant complexity measure of the final algorithm for deciding equality in KAC is the size of the produced automata. In the following the *size* of an automaton is its number of states. In order to give a fair comparison, we will consider the generic algorithms given in the previous subsection, taking as their input a *non-deterministic* automaton, and returning a *deterministic* automaton.

Let us begin by evaluating the size of the automaton produced by the method in [6], given a non-deterministic automaton of size n. As explained above, the states of the constructed transition monoid ($M_{\mathscr{D}}$) are in bijection with some binary relations on Q. There are thus at most 2^{n^2} elements in this monoid. We deduce that the final automaton, whose states are pairs of subsets of $M_{\mathscr{D}}$ has at most $2^{2^{n^2}} \times 2^{2^{n^2}} = 2^{2^{n^2+1}}$ states.

Now with our deterministic construction, the states belong to the set $\mathcal{P}(Q) \times \mathcal{R}^\star(Q)$. Since $\mathcal{R}^\star(Q)$ is the set of reflexive (transitive) relations on Q, we know that $\mathcal{R}^\star(Q)$ has less than $2^{n\times(n-1)}$ elements. Hence we have $|\mathcal{P}(Q) \times \mathcal{R}^\star(Q)| \leqslant 2^n \times 2^{n\times(n-1)} = 2^{n^2}$, which is significantly smaller than the $2^{2^{n^2+1}}$ states we get with the other construction.

2.5.3 A polynomial-space algorithm

The above upper-bound on the number of states of the automata produced by the presented construction allows us to show that the problem of checking equivalence in KAC is PSpace-complete.

Consider PSpace-hardness first. Using the language-theoretic characterisation from Ésik et al. [6, 26], KAC is easily shown to be a conservative extension of KA. Indeed, for any regular expression e, we have $\mathbf{e} = e$ and ${}^{\triangleleft}[\![\mathbf{e}]\!] = [\![e]\!]$, so that for two regular expressions e, f, we have $\mathrm{KAC} \vdash e = f$ iff ${}^{\triangleleft}[\![\mathbf{e}]\!] = {}^{\triangleleft}[\![\mathbf{f}]\!]$ iff $[\![e]\!] = [\![f]\!]$ iff $\mathrm{KA} \vdash e = f$. This is sufficient to conclude since language equivalence of regular expressions is known to be PSpace-complete [46].

Now recall Proposition 1.29, stating that the equivalence of two deterministic automata $\mathscr{A}$ and $\mathscr{B}$ is in LogSpace. The algorithm to show that relies on the fact that $\mathscr{A}$ and $\mathscr{B}$ are different if and only if there is a word w in the difference of $L(\mathscr{A})$ and $L(\mathscr{B})$ such that $|w| \leqslant |\mathscr{A}| \times |\mathscr{B}|$. With that in mind, we can give a non-deterministic algorithm, by simulating a computation in both automata with a letter chosen non-deterministically at each step, with a counter to stop us at size $|\mathscr{A}| \times |\mathscr{B}|$. The resulting algorithm will only have to store the counter of size $\log(|\mathscr{A}| \times |\mathscr{B}|)$ and the two current states.

For deciding KAC, the first step is to compute $\mathbf{e}$ and $\mathbf{f}$ from the regular expressions with converse e and f. It is obvious that such a transformation can be done in linear time and space, by a single sweep of both e and f. Then we have to build automata for $\mathbf{e}$ and $\mathbf{f}$. Once again this is a very light operation: if one considers for instance the position automaton (also called Glushkov's construction [30]), we obtain automata of respective sizes $n = |\mathbf{e}| + 1 = |e| + 1$ and $m = |\mathbf{f}| + 1 = |f| + 1$, where $|\cdot|$ denotes the number of variable leaves of a regular expression (possibly with converse).

Our construction then produces closed automata of size at most 2^{n^2} and 2^{m^2}, so that the non-deterministic algorithm to check their equivalence needs to scan all words of size smaller than by $2^{n^2} \times 2^{m^2} = 2^{n^2+m^2}$. The counter used to bound the recursion depth can thus be stored in polynomial space ($n^2 + m^2$). It is worth mentioning here that with the automata constructed in [6], the counter would have size $2^{n^2+1} + 2^{m^2+1}$ which is not a polynomial.

Now the last two important things to worry about are the representation of the states of the closure automata, in particular their "history" component, and the way to compute their transition function. Let us focus on the automaton for $\mathbf{e}$ and let Q be the set of states of the Glushkov automaton built out of it.

input : Two regular expressions with converse $e, f \in \mathrm{Reg}^{\smile}\langle X \rangle$
output: A Boolean, saying whether or not $\mathrm{KAC} \vdash e = f$.

$\mathscr{A}_1 = \langle Q_1, \mathbf{X}, I_1, \mathscr{T}_1, \Delta_1 \rangle \leftarrow$ Glushkov's automaton recognising $[\![\mathbf{e}]\!]$;
$\mathscr{A}_2 = \langle Q_2, \mathbf{X}, I_2, \mathscr{T}_2, \Delta_2 \rangle \leftarrow$ Glushkov's automaton recognising $[\![\mathbf{f}]\!]$;
$N \leftarrow (2^{(|e|+1)^2} \times 2^{(|f|+1)^2})$; /* N gets a value $\geqslant |{}^{\triangleleft}\mathscr{A}_1| \cdot |{}^{\triangleleft}\mathscr{A}_2|$ */
$\langle \langle P_1, R_1 \rangle, \langle P_2, R_2 \rangle \rangle \leftarrow \langle \langle I_1, \mathrm{Id}_{Q_1} \rangle, \langle I_2, \mathrm{Id}_{Q_1} \rangle \rangle$;
while $N > 0$ **do**
  $N \leftarrow N - 1$; /* N bounds the recursion depth */
  $f_1 \leftarrow$ is_empty$(P_1 \cap \mathscr{T}_1)$;
  $f_2 \leftarrow$ is_empty$(P_2 \cap \mathscr{T}_2)$;
  if $f_1 = f_2$ **then**
    $x \leftarrow$random$(\mathbf{X})$; /* Non-deterministic choice */
    $\langle R_1, R_2 \rangle \leftarrow \langle h_x(R_1), h_x(R_2) \rangle$;
    $\langle P_1, P_2 \rangle \leftarrow \langle P_1 \cdot \Delta_1(x) \cdot R_1, P_2 \cdot \Delta_2(x) \cdot R_2 \rangle$;
  else
    return false; /* A difference appeared for some word, $e \neq f$ */
  end
end
return true; /* There was no difference, KAC $\vdash e = f$ */

Algorithm 1: A PSpace algorithm for KAC

- States are pairs of a set of states in Q and a binary relation (set of pairs) over Q. Such a pair can be stored in polynomial space (recall that $|Q| = n = |e| + 1$).
- For computing the transition function, the image of a pair $\langle \{q_1, \cdots, q_k\}, R \rangle$ (with $R \subseteq Q^2$) by a letter $x \in \mathbf{X}$ is done in two steps: first the relation becomes $R' = h_x(R) := (\Delta(x') \cdot R \cdot \Delta(x))^\star$, then the set of states becomes

$$\{q \mid \exists i, 1 \leqslant i \leqslant k : \langle q_i, q \rangle \in \Delta(x) \cdot R'\}.$$

  Those computations take place in PSpace. (The composition of two relations on Q can be performed in space $\mathcal{O}\left(|Q|^2\right)$, and the same holds for the reflexive and transitive closure of a relation R by building the powers $(R + \mathrm{Id}_Q)^{2^k}$ and keeping a copy of the previous iteration to stop when the fixed-point is reached.)

Summing up, we obtain Algorithm 1, which is PSpace.

2.5.4 Time-efficient algorithms

While the previous algorithm matches the theoretical complexity of the problem, it cannot be used in practice. Indeed, it systematically requires exponential time (except when there exists a small counter-example to the starting equation). This is similar to the case of regular expressions without converse (i.e., Kleene algebra), where one usually implements algorithms that are not PSpace but that require less than exponential time in most cases.

Such algorithms include the standard algorithm by Hopcroft and Karp [32], antichain-based algorithms [62, 1, 24], and more recent algorithms relying on "bisimulations up to congruence" [9]. We show how to exploit the latter technique with the automata produced by the presented construction.

To apply this technique, one needs to work with a single non-deterministic automaton, to be determinised by the powerset construction. The global decision procedure for deciding $\mathrm{KAC} \vdash e = f$ still starts by constructing two automata recognising the languages $[\![\mathbf{e}]\!]$ and $[\![\mathbf{f}]\!]$ (as in the first two lines in Algorithm 1). Then one can either consider the disjoint union

of their closures, or take their union and build the closure afterwards. The first approach makes it possible to use the algorithms from [9] off-the-shelf. A drawback is that depending on the constructions used to build the automata for $[\![\mathbf{e}]\!]$ and $[\![\mathbf{f}]\!]$, it might be natural that they share some states (e.g., when using Antimirov's partial derivatives [4]), and we loose this sharing when computing their closures.

Here we describe the second approach: this enables some additional optimisations, and the above drawback disappears as one can always take an overlapping union. (Note however that if the automata were actually disjoint, taking the closure after the union potentially results in bigger automata: the history part—$\mathcal{R}^\star(Q)$—being shared, parts of one of the underlying automata can be duplicated just because of the history related to the other automaton.) We thus assume a non-deterministic automaton $\langle Q, \mathbf{X}, \mathscr{T}, \Delta\rangle$ with two sets of initial states $I_{\mathbf{e}}$, $I_{\mathbf{f}}$ such that $[\![\mathbf{e}]\!] = L(\langle Q, \mathbf{X}, I_{\mathbf{e}}, \mathscr{T}, \Delta\rangle)$ and $[\![\mathbf{f}]\!] = L(\langle Q, \mathbf{X}, I_{\mathbf{f}}, \mathscr{T}, \Delta\rangle)$.

We thus need an algorithm for checking whether $\langle I_{\mathbf{e}}, \mathrm{Id}_Q\rangle$ and $\langle I_{\mathbf{f}}, \mathrm{Id}_Q\rangle$ are equivalent in the closure of $\langle Q, \mathbf{X}, \mathscr{T}, \Delta\rangle$. Due to the shape of the determinisation of this automaton, we consider *stratified relations*, i.e., relations indexed by histories.

Definition 2.27 (Stratified relation)**.**
A *stratified relation* $\mathcal{R}$ is a function from histories in $\mathcal{R}^\star(Q)$ to relations on sets of states:

$$\mathcal{R}\colon \mathcal{R}^\star(Q) \to \mathcal{R}el\,\langle \mathcal{P}(Q)\rangle$$

We write $P\,\mathcal{R}_R\,P'$ for $\langle P, P'\rangle \in \mathcal{R}(R)$. ∗

One can then define an appropriate notion of (stratified) bisimulation:

Definition 2.28 (Progression, Bisimulation)**.**
Given two stratified relations $\mathcal{R}, \mathcal{R}'$, we say that $\mathcal{R}$ *progresses to* $\mathcal{R}'$, denoted $\mathcal{R} \rightarrowtail \mathcal{R}'$, if whenever $P\,\mathcal{R}_R\,P'$ then

1. $P \cap \mathscr{T} = \emptyset$ if and only if $P' \cap \mathscr{T} = \emptyset$ and
2. for all $x \in \mathbf{X}$, $(P \cdot \Delta(x) \cdot h_x(R))\,\mathcal{R}'_{h_x(R)}\,(P' \cdot \Delta(x) \cdot h_x(R))$.

A *bisimulation* is a stratified relation $\mathcal{R}$ such that $\mathcal{R} \rightarrowtail \mathcal{R}$. ∗

Proposition 2.29 (Coinduction)**.** *The languages* ${}^{\triangleleft}[\![\mathbf{e}]\!]$ *and* ${}^{\triangleleft}[\![\mathbf{f}]\!]$ *are equivalent if and only if there exists a bisimulation* $\mathcal{R}$ *such that* $I_{\mathbf{e}}\,\mathcal{R}_{Id_Q}\,I_{\mathbf{f}}$. ■

Proof. Simple adaptation of the same result in [9], to work with stratified relations. □

Accordingly, we obtain Algorithm 2 (where we assume g to be the identity function, for now). This algorithm works as follows: the variable $\mathcal{R}$ contains a relation which is a bisimulation candidate and the variable L contains a queue of triples $\langle R, P, Q\rangle$ that remain to be processed (R being a history, and P, Q being sets of states). To process such a pair, one first checks whether it already belongs to the bisimulation candidate: in that case, the pair can be skipped since it was already processed. Otherwise, one checks that both sets are either accepting or non-accepting (line 8), and one adds all derivatives of the pair to L (line 9). The triple $\langle R, P, Q\rangle$ is finally added to the bisimulation candidate (line 10), and we proceed with the remainder of the queue. When the queue L becomes empty, then $\mathcal{R}$ is a bisimulation thanks to the main loop invariant (line 5—recall that for now, g is the identity function), so that the starting expressions are equivalent.

This algorithm can be enhanced by exploiting *up-to techniques* [58, 53]: an up-to technique is a function g on (stratified) relations such that any relation $\mathcal{R}$ satisfying $\mathcal{R} \rightarrowtail g(\mathcal{R})$ is contained in a bisimulation. Intuitively, such relations, that are not necessarily bisimulations, are constrained enough to contain only language equivalent pairs.

input : Two regular expressions with converse $e, f \in \mathrm{Reg}^{\smile}\langle X \rangle$
output: A Boolean, saying whether or not $\mathrm{KAC} \vdash e = f$.

```
1  ⟨Q, X, I_e, I_f, 𝒯, Δ⟩ ← Antimirov's automaton recognising [[e]] and [[f]];
2  L ← {⟨Id_Q, I_e, I_f⟩} ;                    /* Set of elements to be processed */
3  R ← (_ ↦ ∅) ;   /* Stratified relation meant to become a bisimulation */
4  while L ≠ ∅ do
5  |  // R ↣ g(R) ∪ L
6  |  Pick ⟨R, P, P'⟩ from L;
7  |  if ⟨P, P'⟩ ∉ g(R)(R) then
8  |  |  if is_empty(P ∩ 𝒯) = is_empty(P' ∩ 𝒯) then
9  |  |  |  foreach x ∈ X do Add (h_x(R), P · Δ(x) · h_x(R), P' · Δ(x) · h_x(R)) to L
   |  |  |   ;
10 |  |  |  Add (P, P') to R(R);
11 |  |  else
12 |  |  |  return false ;   /* A difference appeared for some word, e ≠ f */
13 end
14 return true ;   /* There was no difference, R is a bisimulation up to g,
      KAC ⊢ e = f */
```

Algorithm 2: Time-efficient algorithms for KAC, g may range over various up-to techniques.

Examples of such up-to techniques include:

1. *equivalence closure* (in the present context of stratified relations, the function e associating to a stratified relation $\mathcal{R}$ the stratified relation $e(\mathcal{R})$ mapping any history R to the smallest equivalence relation that contains $\mathcal{R}(R)$). This technique is implicitly used in the algorithm by Hopcroft and Karp [32], via a disjoint-set forest data structure: we get their algorithm by choosing $g = e$ in Algorithm 2.

2. *congruence closure*: the function c mapping any history R to the smallest equivalence relation S that contains $\mathcal{R}(R)$ and that satisfies the following rule.

$$\frac{P_i \; S \; P'_i, \; i = 1, 2}{(P_1 \cup P_2) \; S \; (P'_1 \cup P'_2)}$$

   This is the key up-to technique introduced in [9]; it allows one to avoid exploring large parts of the automaton, and to stop much earlier than with other algorithms (some sets of states that are accessible through the power-set construction need not be visited at all.)

Proposition 2.30. *The above functions e and c are valid up-to techniques in the present setting.* ■

Proof. Following the same arguments as in [9]. In particular, it holds that $\mathcal{R} \rightarrowtail e(\mathcal{R})$ (resp. $\mathcal{R} \rightarrowtail c(\mathcal{R})$) entails $e(\mathcal{R}) \rightarrowtail e(\mathcal{R})$ (resp. $c(\mathcal{R}) \rightarrowtail c(\mathcal{R})$). □

The resulting algorithms (Algorithm 2 with g instantiated with either e or c) require exponential time (and space) in worst case, as the final bisimulation candidate, $\mathcal{R}$, can be exponentially large. However in practice, like in the simpler case of Kleene algebra without converse, those worst cases are hard to reach, so that such algorithms can usually cope with expressions with up to a thousand of nodes [9].

2.6 Conclusion

Building on the work of Bernátsky, Bloom, Ésik and Stefanescu, we gave new and more efficient algorithms to decide the theory KAC. These algorithms rely on a more compact construction for the closure of an automaton. The first one (Algorithm 1) is PSpace, which allowed us to show that the equational theory of KAC is PSpace-complete. The other ones (the two main instances of Algorithm 2) are not PSpace, but they seem to work well in practice; they are variants of the standard Hopcroft and Karp's algorithm [32], and of its recent optimisation using bisimulations up to congruence [9].

As an exercise, we have implemented and tested the various constructions and algorithms in an OCaml program which is available online [12].

To prove the correctness of the main automata construction, we used the family of regular languages $\Gamma(w)$ (corresponding to $G(w^\smile)$ in [6]). We established the main properties of this family using a proper finite automata characterisation. Moreover, this family allowed us to reformulate the proof of the completeness of the reduction from equality in $\mathcal{R}el$ to equivalence of closed automata (implication (2.8) from the introduction).

To continue this work, we would like to implement one of the presented algorithms in the proof assistant Coq, as a tactic to automatically prove the equalities in KAC—as it has already been done for the theories KA [11] and KAT [52]. The simplifications we propose in this paper give us hope that such a task is feasible. The main difficulty certainly lies in the formalisation of the completeness proof of KAC (implication (2.9) from the introduction), whose key step consists in proving that for all expression $\mathbf{e} \in \mathrm{Reg}\,\langle \mathbf{X} \rangle$, there exists a proof of $\mathbf{e} = {}^{\triangleleft}\mathbf{e}$ in KAC (where ${}^{\triangleleft}\mathbf{e}$ is a regular expression for the regular language ${}^{\triangleleft}[\![\mathbf{e}]\!]$). This is established in [26], but the proof uses yet another automaton construction for the closure, which is even more complicated than the one used in [6], and which seems quite difficult to formalise in Coq. We hope to find an alternative completeness proof, by exploiting our simpler construction.

Third chapter

A Kleene theorem for graph languages

"The Tao produced One; One produced Two; Two produced Three; Three produced All things."

— Lao Tzu, *Tao Te Ching.*

3.1 Introduction

Petri Automata are automata based on Petri nets, whose operational semantics is designed to recognise sets of graphs. We introduced them in [16] to study Kleene allegory expressions: terms built from a finite alphabet of variables, with the constants 0 and 1, the unary operators converse and Kleene star, and the binary operators union, composition and intersection. To any such expression, one can associate a set of graphs such that two expressions are universally equivalent when interpreted as binary relations if and only if their associated sets of graphs are equal. The construction of these sets of graphs from expressions is a direct adaptation of the independent developments of Freyd and Scedrov [27] and Andréka and Bredikhin [2]. A small extension of this study will be the object of Chapter 4.

Independently of this original intent, these models of expressions and Petri automata are relatively simple to define, and capture an interesting class of graph languages. For these reasons, we purposely give in this chapter a presentation of these notions without mentioning the original goal of studying relational equivalence.

For the usual notion of finite state automata, the well known Kleene Theorem states that the languages recognisable by automata are exactly those specifiable by regular expressions. In this chapter we provide a similar theorem for Petri automata and what we call graph expressions.

3.2 Regular and recognisable sets of graphs

We start by describing the graphs we consider, before proceeding to the definition of regular sets of such graphs. Then we present Petri automata and the sets of graphs they recognise.

3.2.1 Graphs

We introduce a few definitions and results about graphs. Unless otherwise stated, all graphs are directed, acyclic and their edges are labelled with some fixed alphabet Σ. The relation $\equiv$ relates isomorphic graphs. We will generally consider graphs up-to $\equiv$.

DAGs and trees Given a directed acyclic graph (DAG) $G = \langle V, E \rangle$, we may define its *minimum* $\min G$ (resp. *maximum* $\max G$) to be the set of vertices in V with no incoming (resp. outgoing) edge. A vertex v is *reachable* (respectively *co-reachable*) from another vertex v' if there is a path in G from v' to v (resp. from v to v'). A sub-graph C of G is *connected* if there is a non-directed path between any two vertices. It is called a *connected component* if there does not exist a connected sub-graph of G containing C, apart from C itself.

A *tree* is a graph $T = \langle V, E \rangle$ such that either $\min T$ or $\max T$ contains a single node, called the root, and for any two nodes $x, y \in V$ there exists at most one path from x to y. If $\min T$ is a singleton, then T is a *top-down tree*, otherwise it is a *bottom-up tree*. A tree is said to be proper if its root has degree one and if it does not contain a node with exactly

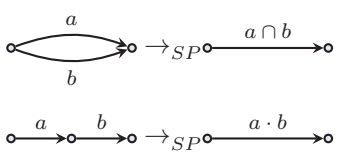


Figure 3.1: The SP-rewriting system

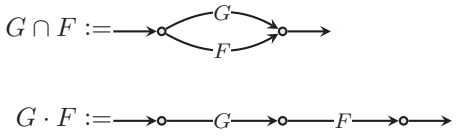


Figure 3.2: Elementary graph constructions

one incoming edge and one outgoing edge. There is only a finite number of proper trees with leaves chosen from a finite set.

Two-Terminal Series Parallel graphs We use the *SP-rewriting system*, enriched with labels, as presented in Figure 3.1. (Note that the second rule can only be applied if the middle vertex on the left-hand side has no other adjacent edge.) We write $G \to_{SP}^{\star} F$ if G reduces to F in a finite number of SP-reduction steps. Valdes et al. [61] showed that that system (without the labels) has the Church-Rosser property. This property can be extended to labelled graphs without difficulty, modulo the congruence generated by the associativity of $\cdot$ and the associativity and commutativity of $\cap$.

A graph G is *Two Terminal Series Parallel* (TTSP) if $G \to_{SP}^{\star} \overset{e}{\longrightarrow}$, for some term e. This term is built out of letters from Σ and the operators $\cdot$ and $\cap$, and is called the *term representation* of G (written $\mathscr{W}(G)$). Any TTSP graph has a single source (vertex of in-degree zero) called its *input* and a single sink (vertex of out-degree zero) called its *output*. When convenient, we will represent such a graph as $\langle V, E, \iota, f\rangle$, where $\langle V, E\rangle$ is the TTSP graph, and ι and f are its input and output.

The class of TTSP graph is stable under sequential and parallel composition. The *sequential composition* (written $\_ \cdot \_$) of two TTSP graphs with disjoint sets of vertices can be performed by identifying the output of the first graph and the input of the second one. Their *parallel composition* (written $\_ \cap \_$) consists in merging their inputs and merging their outputs. See Figure 3.2 for a graphical description of these constructions.

Technical result The following lemma will be instrumental later on. Let $T = \langle V_T, E_T\rangle$ be a proper unlabelled top down tree with root r and set of leaves $F \subseteq V_T$, and $G = \langle V, E\rangle$ be a connected DAG. Let $\varphi : F \rightharpoonup V$ be a partial function defined on $F' \subseteq F$. The *gluing* of T and G along φ is the graph

$$T \cdot_{\varphi} G := \langle V_T \cup V, E_T \cup E \cup \{\langle f, \varphi(f)\rangle\}\rangle .$$

Lemma 3.1. *If $T \cdot_{\varphi} G \to_{SP}^{\star} T'$ and if T' is a tree, then there is a node c in G such that for every $\langle f, M\rangle \in F' \times \max G$ every path from $\varphi(f)$ to M in G visits c.* ■

This lemma makes intuitive sense. The maximal elements of G will remain in T' as leaves, and because G is connected there will be a subtree of T' whose leaves are exactly $\max G$. The root of this tree must be a node accessible from F', hence coming from G. Because paths are preserved during SP-rewriting, this vertex has the desired property.

However, writing a formal proof of this lemma proved to be a bit of a challenge. We present one here, and apologise for its rather tedious nature...

Proof. We begin by defining a family of sets of vertices V_i, a family of functions $[]_i$ and a family of relations $\to_i$, allowing to trace vertices and paths during the SP-reduction. $\langle V_i, \to_i\rangle$ is meant to be the graph obtained from $T \cdot_{\varphi} G$ after i steps of SP-reduction. $[x]_i \in V_i$ is

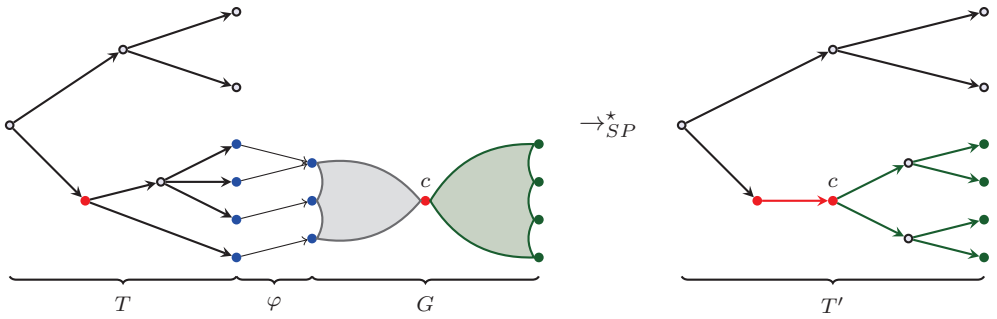


Figure 3.3: Illustration of Lemma 3.1

either x if x was not deleted during the reduction, or a vertex x was merged with at a certain step.

Initialisation $V_0 := V_T \cup V$; $\to_0 := E_T \cup E \cup \{\langle f, \varphi(f)\rangle\}$; $[x]_0 := x$.

At step $i+1$ Let us examine what rule of Figure 3.1 was executed:

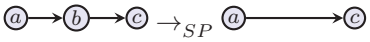
- if this step uses the first rule (parallel reduction), then $V_{i+1} = V_i$, $\to_{i+1} = \to_i$, and $[x]_{i+1} = [x]_i$;
- otherwise, let a, b, c be the three vertices in the left hand side of the rewriting rule, then $V_{i+1} = V_i \setminus \{b\}$, $\to_{i+1} = \to_i \setminus \{\langle a, b\rangle, \langle b, c\rangle\} \cup \{\langle a, c\rangle\}$, and for every $x \in V_0$ if $[x]_i = b$ then $[x]_{i+1} = c$ otherwise $[x]_{i+1} = [x]_i$.

Notice that $V_{i+1} \subseteq V_i$, and if $x \in V_i$ then $[x]_i = x$. It is also immediate to check that the maximal vertices of G, who do not have outgoing edges in $T \cdot_\varphi G$, are never deleted. This means that for every i we have $\max G \subseteq V_i$, and thus if $m \in \max G$, then $[m]_i = m$.

We need to establish an intermediary result: for every pair of vertices $x, y \in V_0$ and every step i we have:

$$\text{if } x, y \in V_i,\ x \to_i y \Rightarrow x \to_0^+ y \tag{3.1}$$
$$x \to_0^\star [x]_i \tag{3.2}$$
$$\text{if } x \in V_i,\ x \to_0 y \Rightarrow [x]_i \to_i [y]_i \tag{3.3}$$
$$\text{if } x \notin V_i,\ x \to_0 y \Rightarrow [x]_i = [y]_i \tag{3.4}$$

We prove this by induction on i. The initialisation being trivial, we only show the induction step, and only in the case where the reduction step is of the second kind. Hence we suppose that step $i+1$ is:



1. For (3.1) we simply check that $\to_{i+1} \subseteq \to_i \cup \to_i^2$.
2. If $[x]_i \neq b$ then $[x]_{i+1} = [x]_i$, so the induction hypothesis (IH) allows to conclude. If on the other hand $[x]_i = b$, by IH we know that $x \to_0^\star b$, and because $b \to_i c$ and (3.1) we know that $b \to_0^+ c$. Hence we obtain $x \to_0^\star c = [x]_{i+1}$.
3. If $x \to_0 y$ and $x \in V_{i+1}$, then $x \in V_i$ so the IH applies and we get $[x]_i \to_i [y]_i$. We also know that $[x]_{i+1} = [x]_i$. Either $[y]_i = b$ or $[y]_{i+1} = [y]_i$, and in both cases $[x]_{i+1} \to_{i+1} [y]_{i+1}$.

4. If $x \to_0 y$ and $x \notin V_{i+1}$, then either $x \notin V_i$ or $x = b$. In the first case we apply the IH and get $[x]_i = [y]_i$ which means $[x]_{i+1} = [y]_{i+1}$. In the second case the IH gives us $[x]_i \to_i [y]_i$, which imposes that $[y]_i = c$, hence $[y]_{i+1} = c = [x]_{i+1}$.

Furthermore, equations (3.3) and (3.4) allows us to check that every path in $\langle V_0, \to_0 \rangle$ from x to $m \in \max G$ visits $[x]_i$, for every i. This can be proved by induction on paths: if the path is trivial, meaning $x = m$, then $[m]_i = m$; for a path $x \to_0 y \to_0^\star m$ we know that either $x \in V_i$, thus $[x]_i = x$, or $[x]_i = [y]_i$, allowing us to conclude.

We will now prove that if the graph is a tree at step i, then there is a vertex c in $V \cap V_i$ such that for every leaf f in F' we have $[f]_i = c$.

First, the representative of every $f \in F'$ is in G. By construction, f will be deleted because it has exactly one incoming edge and one outgoing edge. By (3.2) its representative must be reachable from f, thus has to be in G.

To prove that they are all sent to the same vertex, we use the fact that trees have no non-oriented cycles. Consider the cycle build by linking two leaves $f, f' \in F'$ first through G (which is possible because G is connected) and then through T (by going back to their most recent ancestor). At each reduction step, if a vertex of the cycle is deleted, then its new representative is another vertex of the cycle: indeed, when a node is deleted it only has two incident edges, and every node of the cycle has two distinct neighbours in the cycle. This means that if the graph is a tree at step i, the cycle must have been reduced to either a single vertex, or two vertices linked by an edge. In the latter case, it must be the case that the most recent common ancestor of f and f' is represented by the source of this edge, and f and f' are represented by its target.

Finally, every path from $\varphi(f)$ to $m \in \max G$ can be extend to a path from f to m, and all of those must visit $[f]_i = c$. □

3.2.2 Regular graph expressions

Definition 3.2 (Regular graph expressions)**.**
Regular graph expressions, or simply expressions in the remaining of this chapter, are terms over the following syntax, where a denotes any letter from Σ:

$$e, f ::= 0 \mid a \mid e \cup f \mid e \cdot f \mid e \cap f \mid e^+.$$

We denote by $\mathrm{GReg}\langle\Sigma\rangle$ the set of expressions over the finite alphabet Σ. ∗

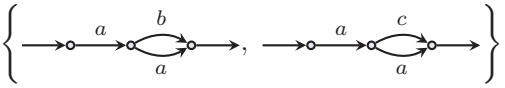
We assign to each expression a set of TTSP graphs, called the graph language of the expression.

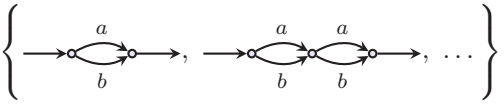
Definition 3.3 (Graph language of an expression, regular sets)**.**
The *graph language* of an expression is defined as follows by structural induction:

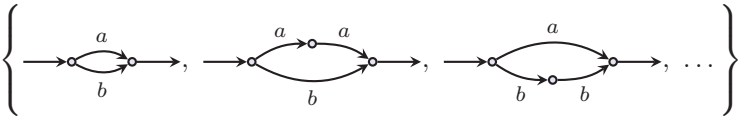
$$\mathscr{G}(a) := \left\{ \xrightarrow{\quad a \quad} \right\} \qquad \mathscr{G}(e \cdot f) := \{E \cdot F \mid E \in \mathscr{G}(e) \text{ and } F \in \mathscr{G}(f)\}$$
$$\mathscr{G}(0) := \emptyset \qquad \mathscr{G}(e \cap f) := \{E \cap F \mid E \in \mathscr{G}(e) \text{ and } F \in \mathscr{G}(f)\}$$
$$\mathscr{G}(e \cup f) := \mathscr{G}(e) \cup \mathscr{G}(f) \qquad \mathscr{G}(e^+) := \{E_1 \cdot \ldots \cdot E_n \mid n > 0,\ \forall i, E_i \in \mathscr{G}(e)\}$$

A set of graphs S is said to be *regular* if there is an expression $e \in \mathrm{GReg}\langle\Sigma\rangle$ such that $\mathscr{G}(e) = S$. ∗

Example 3.4.

$\mathscr{G}(a \cdot ((b \cup c) \cap a)) :=$ 

$\mathscr{G}\left((a \cap b)^+\right) :=$ 

$\mathscr{G}\left(\left(a^+\right) \cap \left(b^+\right)\right) :=$ 

•

Remark. Notice that the graphs produced by this construction are exactly TTSP graphs labelled with Σ.

Remark. If e does not contain the operators 0, $\cup$ nor $\_^+$, then $\mathscr{G}(e)$ is a singleton set. In this case, we may identify $\mathscr{G}(e)$ and the only graph it contains. Notice that if e is such an expression, e is equivalent to $\mathscr{W}(\mathscr{G}(e))$, modulo associativity of $\cdot$ and associativity and commutativity of $\cap$. Conversely, any TTSP graph G is isomorphic to $\mathscr{G}(\mathscr{W}(G))$.

3.2.3 Petri automata

A Petri automaton is essentially a safe Petri net where the arcs coming out of transitions are labelled by letters from Σ.

Definition 3.5 (Petri Automaton)**.**
A *Petri automaton* $\mathscr{A}$ over the alphabet Σ is a tuple $\langle P, \mathscr{T}, \iota \rangle$ where:

- P is a finite set of *places*,
- $\mathscr{T} \subseteq \mathcal{P}(P) \times \mathcal{P}(\Sigma \times P)$ is a set of *transitions*,
- $\iota \in P$ is the *initial place* of the automaton.

For each transition $t = \langle {}^\triangleright t, {}^\triangleleft t \rangle \in \mathscr{T}$, ${}^\triangleright t$ is assumed to be non-empty; ${}^\triangleright t \subseteq P$ is the *input of* t; and ${}^\triangleleft t \subseteq \Sigma \times P$ is the *output of* t. Transitions with empty outputs are called *final*, and transitions with input $\{\iota\}$ are called *initial*. ∗

We write $\pi_2({}^\triangleleft t) := \{p \mid \exists a, \langle a, p \rangle \in {}^\triangleleft t\}$ for the set of places appearing in the output of t. We will add a few constraints on this definition along the way, but we need more definitions to state them. An example of such an automaton is described in Figure 3.4. The graphical representation used here draws round vertices for places and rectangular vertices for transitions, with the incoming and outgoing arcs to and from the transition corresponding respectively to the inputs and outputs of said transition. The initial place is denoted by an unmarked incoming arc.

Runs and reachable states. We define the operational semantics of Petri automata. Let us fix a Petri automaton $\mathscr{A} = \langle P, \mathscr{T}, \iota \rangle$ for the remainder of this section. A *state* of this automaton is a set of places. In a given state $S \subseteq P$, a transition $t = \langle {}^\triangleright t, {}^\triangleleft t \rangle$ is *enabled* if ${}^\triangleright t \subseteq S$. In that case, we may fire t, leading to a new state $S' = (S \setminus {}^\triangleright t) \cup \pi_2({}^\triangleleft t)$. This will be denoted in the following by $\mathscr{A} \vdash t : S \to S'$. We extend this notation to sequences of transitions in the natural way:

$$\frac{\mathscr{A} \vdash t_1 : S_0 \to S_1, \quad \mathscr{A} \vdash t_2; \ldots; t_n : S_1 \to S_n}{\mathscr{A} \vdash t_1; t_2; \ldots; t_n : S_0 \to S_n}$$

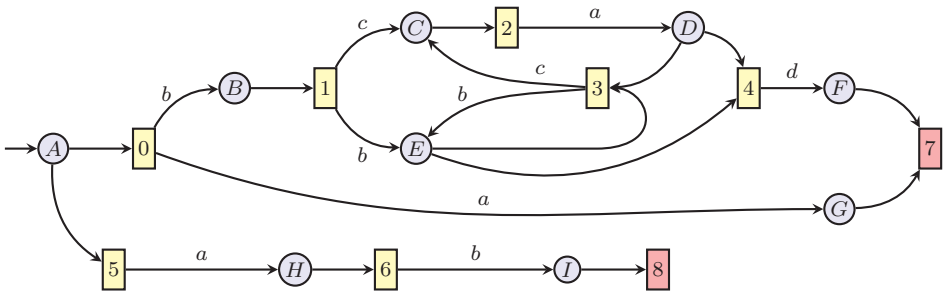


Figure 3.4: A Petri automaton.

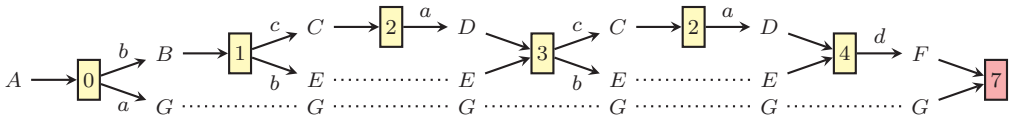


Figure 3.5: An accepting run in the automaton from Figure 3.4.

In that case we say that $\langle S_0, t_1; t_2; \ldots; t_n, S_n\rangle$ is a *valid run*, or simply run, from S_0 to S_n. If $S_0 = \{\iota\}$ then the run is *initial* and if $S_n = \emptyset$ it is *final*. A run that is both initial and final is called *accepting*. A state S is reachable in $\mathscr{A}$ if there is an initial run leading to S.

We write $\mathrm{Run}_{\mathscr{A}}$ for the set of runs of an automaton $\mathscr{A}$, and $\mathrm{Run}_{\mathscr{A}}(A, B)$ for the set of runs from state A to state B in $\mathscr{A}$. The set of its accepting runs is then equal to $\mathrm{Run}_{\mathscr{A}}(\{\iota\}, \emptyset)$, and written $\mathrm{Run}_{\mathscr{A}}^{acc}$.

Example 3.6 (Accepting run)**.**
The triple $\langle\{A\}, 0; 1; 2; 3; 2; 4; 7, \emptyset\rangle$ is a valid run in the automaton from Figure 3.4. It is easy enough to check that:

$$\begin{array}{ll} \mathscr{A} \vdash 0 : \{A\} \to \{B, G\}; & \mathscr{A} \vdash 1 : \{B, G\} \to \{C, E, G\}; \\ \mathscr{A} \vdash 2 : \{C, E, G\} \to \{D, E, G\}; & \mathscr{A} \vdash 3 : \{D, E, G\} \to \{C, E, G\}; \\ \mathscr{A} \vdash 4 : \{D, E, G\} \to \{F, G\}; & \mathscr{A} \vdash 7 : \{F, G\} \to \emptyset. \end{array}$$

Thus we can prove that $\mathscr{A} \vdash 0; 1; 2; 3; 2; 4; 7 : \{A\} \to \emptyset$. Furthermore, as A is the initial place, this run is accepting. It can be represented graphically as in Figure 3.5. •

We may now state the first constraint we impose on Petri automata.

Constraint 1 (Safety)**.** We impose on every Petri automaton $\mathscr{A}$ that for any states S and S' and every transition t, if S is reachable and $\mathscr{A} \vdash t : S \to S'$, then

$$(S \setminus {}^{\triangleright}t) \cap \pi_2({}^{\triangleleft}t) = \emptyset. \qquad ✠$$

This constraint corresponds to the classic Petri net property of safety, also called one-boundedness.

Remark. Constraint 1 is quite easily decidable: the set of transitions is finite, and because reachable states are subsets of a fixed finite set, there is only finitely many of those. Thus checking the constraint only entails a finite number of tests.

Traces. The graph language of a Petri automaton can be obtained by extracting from every accepting run a graph, called its *trace*. Consider an accepting run $\langle\{\iota\}, t_0; \ldots; t_n, \emptyset\rangle$. Its trace is constructed by creating a vertex k for each transition $t_k = \langle {}^{\triangleright}t_k, {}^{\triangleleft}t_k\rangle$ of the run. We add an edge $\langle k, a, l\rangle$ whenever there is some place q such that $\langle a, q\rangle \in {}^{\triangleleft}t_k$, and t_l is the

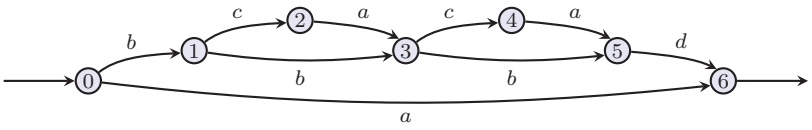


Figure 3.6: Trace of the run from Example 3.6.

first transition after t_k in the run with q among its inputs. The definition we give here is a generalisation for arbitrary valid runs, that coincides with the informal presentation we just gave on accepting runs.

Definition 3.7 (Trace of a run)**.**
Let $R = \langle S, t_0; \ldots; t_n, S' \rangle$ be a run in $\mathscr{A}$. For every k and $p \in \pi_2\left({}^{\triangleleft}t_k\right)$, we define

$$\nu\left(k,p\right) = \left\{l \mid l > k \text{ and } p \in {}^{\triangleright}t_l\right\}.$$

The *trace* of R, denoted by $\mathscr{G}\left(R\right)$, is the graph with vertices $\{0,\ldots,n\} \cup S'$ and the set of edges defined by:

$$E_R = \left\{\langle k,a,l\rangle \mid \langle a,p\rangle \in {}^{\triangleleft}t_k \text{ and } \left(p = l \wedge \nu\left(k,p\right) = \emptyset\right) \vee \left(l = \min\left(\nu\left(k,p\right)\right)\right)\right\}. \qquad *$$

The trace of the run given in Example 3.6 is presented in Figure 3.6. We can now define the language of a Petri automaton:

Definition 3.8 (Language of a Petri automaton, recognisable sets)**.**
The *language of an automaton* $\mathscr{A}$, is the set of traces of accepting runs of $\mathscr{A}$:

$$\mathscr{G}\left(\mathscr{A}\right) := \left\{\mathscr{G}\left(R\right) \mid R \in \mathrm{Run}^{acc}_{\mathscr{A}}\right\}. \qquad *$$

A set of graphs S is called *recognisable* is there is an automaton $\mathscr{A}$ such that $S \equiv \mathscr{G}\left(\mathscr{A}\right)$.

This allows us to state the second constraint we ask of Petri automata.

Constraint 2 (Series Parallel)**.** Every graph $G \in \mathscr{G}\left(\mathscr{A}\right)$ should be a Two Terminal Series Parallel graph. ✠

Despite its infinitary formulation, this property is decidable. In fact, the procedure we describe later on would fail (in finite time) if it is given an automaton violating this constraint, and succeed otherwise: hence it may be used to decide this property. See Section 3.4 for more details.

Because of this constraint, we may restrict further the set of runs we consider:

Definition 3.9 (Proper run)**.**
A run $R = \langle S_0, t_1; \ldots; t_n, S_n \rangle \in \mathrm{Run}_{\mathscr{A}}$ is *proper* if for all $1 \leqslant i \leqslant n$:

$${}^{\triangleleft}t_i = \emptyset \Rightarrow \left(i = n \wedge S_n = \emptyset\right). \qquad *$$

If R is accepting and if $\mathscr{G}\left(R\right)$ is TTSP, then R must be proper, as we prove in the next lemma. As every sub run of a proper run is proper, this means that we may restrict ourselves to proper runs without impacting the set of accepting runs.

Lemma 3.10. *Let $R = \langle \{\iota\}, t_0; \ldots; t_n, \emptyset \rangle$ be an accepting run in $\mathscr{A}$. Then $\mathscr{G}\left(R\right)$ is TTSP only if R is proper.* ■

Proof. For an accepting run, because the last state is empty we know that the run must contain a final transition. Hence we may reformulate the definition of proper as ${}^{\triangleleft}t_n = \emptyset$ and $\forall 0 \leqslant i < n$, ${}^{\triangleleft}t_i \neq \emptyset$.

The proof relies on the following observation: the node i in $\mathscr{G}(R)$ is a sink if and only if ${}^{\triangleleft}t_i = \emptyset$. To prove this, remember that the edges coming out of i in $\mathscr{G}(R)$ are:

$$\{\langle i, a, \min\{l \mid l > i \text{ and } p \in {}^{\triangleright}t_l\}\rangle \mid \forall \langle a, p\rangle \in {}^{\triangleleft}t_i\}.$$

Because the run ends in state $\emptyset$, we know that $\forall \langle a, p\rangle \in {}^{\triangleleft}t_i$ there is a later transition t_j consuming the place p (otherwise p would remain in the last state). This entails that if t_i is not final, then i has a successor in $\mathscr{G}(R)$. On the other hand, if ${}^{\triangleleft}t_i$ is empty, then there cannot be a transition coming out of node i.

As a TTSP graph has a single sink, it can only use a single final transition. Lastly, notice that $\langle i, a, j\rangle \in E_R$ entails $i < j$. This means that the sink can only be the maximal node (for the ordering of natural numbers). □

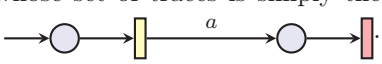
3.2.4 Building automata from expressions

In this section, we show how to produce from any expression $e \in \mathrm{GReg}\langle\Sigma\rangle$ a Petri automaton $\mathscr{A}(e)$ such that:

$$\mathscr{G}(e) \equiv \mathscr{G}(\mathscr{A}(e)).$$

The construction goes by induction, providing automata for the base cases 0 and $a \in \Sigma$, and then showing how to combine two automata $\mathscr{A}_1$ and $\mathscr{A}_2$ to get automata for the languages $\mathscr{G}(\mathscr{A}_1) \cdot \mathscr{G}(\mathscr{A}_2)$, $\mathscr{G}(\mathscr{A}_1) \cap \mathscr{G}(\mathscr{A}_2)$, $\mathscr{G}(\mathscr{A}_1) + \mathscr{G}(\mathscr{A}_2)$, and $\mathscr{G}(\mathscr{A}_1)^+$, thus proving that recognisable languages are closed under union, iteration and sequential and parallel products.

The automaton for the base cases. We need the automaton for 0 to have no accepting run. A good choice for $\mathscr{A}(0)$ is then the automaton:  .

Let $a \in \Sigma$. We want an automaton whose set of traces is simply the graph $\mathscr{G}(a)$. This yields the following automaton $\mathscr{A}(a)$: 

.

Sequence, union, and iteration of automata. Let $\mathscr{A}_1 = \langle P_1, \mathscr{T}_1, \iota_1\rangle$ and $\mathscr{A}_2 = \langle P_2, \mathscr{T}_2, \iota_2\rangle$ be two Petri automata with disjoint sets of places.

Computing the "union" of $\mathscr{A}_1$ and $\mathscr{A}_2$ is quite easy. We simply put the two automata side by side, add a new initial place ι, and for every initial transition $\langle\{\iota_i\}, {}^{\triangleleft}t\rangle$ we add a new transition $\langle\{\iota\}, {}^{\triangleleft}t\rangle$. Formally:

$$\mathscr{A}_1 \cup \mathscr{A}_2 := \langle P_1 \cup P_2 \cup \{\iota\}, \mathscr{T}_1 \cup \mathscr{T}_2 \cup \{\langle\{\iota\}, {}^{\triangleleft}t\rangle \mid \langle\{\iota_i\}, {}^{\triangleleft}t\rangle \in \mathscr{T}_i, i \in \{1,2\}\}, \iota\rangle.$$

This means the transitions in this automaton follow from the set of rules:

$$\frac{\mathscr{A}_i \vdash t : S \to T}{\mathscr{A}_1 \cup \mathscr{A}_2 \vdash t : S \to T}\ i \in 1,2 \qquad \frac{\mathscr{A}_i \vdash \langle\{\iota_i\}, S\rangle : \{\iota_i\} \to S}{\mathscr{A}_1 \cup \mathscr{A}_2 \vdash \langle\{\iota\}, S\rangle : \{\iota\} \to S}\ i \in 1,2$$

Lemma 3.11. $\mathscr{G}(\mathscr{A}_1 \cup \mathscr{A}_2) = \mathscr{G}(\mathscr{A}_1) \cup \mathscr{G}(\mathscr{A}_2)$. ■

Proof. Let $G \in \mathscr{G}(\mathscr{A}_1) \cup \mathscr{G}(\mathscr{A}_2)$. There is an accepting run $R = \langle\{\iota_i\}, t_0; t_1 \dots t_n; \emptyset\rangle$ in $\mathscr{A}_i$, for either $i = 1$ or $i = 2$, such that $G \equiv \mathscr{G}(R)$. By definition of a valid run, we know that there is a state S such that $\mathscr{A}_i \vdash t_0 : \{\iota_i\} \to S$ and $\mathscr{A}_i \vdash t_1; \dots; t_n : S \to \emptyset$. Because $\mathscr{A}_1 \cup \mathscr{A}_2$ contains in particular all places and transitions of $\mathscr{A}_i$, we can deduce that $\mathscr{A}_1 \cup \mathscr{A}_2 \vdash t_1; \dots; t_n : S \to \emptyset$. Furthermore $\mathscr{A}_i \vdash t_0 : \{\iota_i\} \to S$ entails $t_0 = \langle\{\iota_i\}, {}^{\triangleleft}t_0\rangle$, thus in $\mathscr{A}_1 \cup \mathscr{A}_2$ we have the transition $t_0{}' = \langle\{\iota\}, {}^{\triangleleft}t_0\rangle$. Hence: $\mathscr{A}_1 \cup \mathscr{A}_2 \vdash t_0{}' : \{\iota\} \to S$. This

proves that $R' = \langle \{\iota\}, t_0'; t_1 \dots t_n; \emptyset \rangle$ is an accepting run in $\mathscr{A}_1 \cup \mathscr{A}_2$. As $\mathscr{G}(R') = \mathscr{G}(R)$, we have proved that $G \in \mathscr{G}(\mathscr{A}_1 \cup \mathscr{A}_2)$.

Now take $G \in \mathscr{G}(\mathscr{A}_1 \cup \mathscr{A}_2)$, and $R = \langle \{\iota\}, t_0; t_1 \dots t_n; \emptyset \rangle$ such that $R \equiv \mathscr{G}(G)$. Necessarily, t_0 is of the shape $\langle \{\iota\}, {}^{\triangleleft}t_0 \rangle$, and thus was produced from ${t_0}' = \langle \{\iota_i\}, {}^{\triangleleft}t_0 \rangle \in \mathscr{T}_i$, for either $i = 1$ or $i = 2$. If S_0 is the state reached after the first transition, meaning $\mathscr{A}_1 \cup \mathscr{A}_2 \vdash t_0 : \{\iota_i\} \to S_0$, it follows that $S_0 \subseteq P_i$. Because we assumed $P_1 \cap P_2 = \emptyset$, it is straightforward that $\mathscr{A}_1 \cup \mathscr{A}_2 \vdash t : S \to T$ and $S \subseteq P_i$ entails $\mathscr{A}_i \vdash t : S \to T$ and $T \subseteq P_i$. This result extends to sequences of transitions, allowing us to check that because $\mathscr{A}_1 \cup \mathscr{A}_2 \vdash t_1; \dots; t_n : S_0 \to \emptyset$ and $S_0 \subseteq P_i$ we have $\mathscr{A}_i \vdash t_1; \dots; t_n : S_0 \to \emptyset$. Thus $R' = \langle \{\iota_i\}, t_0'; t_1 \dots t_n; \emptyset \rangle$ is an accepting run in $\mathscr{A}_i$, and as $\mathscr{G}(R) = \mathscr{G}(R')$, we get $G \in \mathscr{G}(\mathscr{A}_i) \subseteq \mathscr{G}(\mathscr{A}_1) \cup \mathscr{G}(\mathscr{A}_2)$. □

For the sequence we want that every accepting run R_1 in $\mathscr{A}_1$ can be followed by any accepting run R_2 in $\mathscr{A}_2$. Because of Lemma 3.10, we know that R_1 must end with a final transition, and the definition of accepting run imposes that R_2 starts with an initial transition. Thus it suffices to put $\mathscr{A}_1$ in front of $\mathscr{A}_2$, declare ι_1 initial, remove the final transitions $\langle {}^{\triangleright}t, \emptyset \rangle$ in $\mathscr{A}_1$ and replace them with $\langle {}^{\triangleright}t, {}^{\triangleleft}t \rangle$, for every initial transition $\langle \{\iota_2\}, {}^{\triangleleft}t \rangle$ in $\mathscr{A}_2$. This yields the following definition:

$$\mathscr{A}_1 \cdot \mathscr{A}_2 := \langle P_1 \cup P_2, \mathscr{T}, \iota_1 \rangle,$$

where $\mathscr{T} := (\mathscr{T}_1 \setminus \mathcal{P}(P_1) \times \{\emptyset\}) \cup \mathscr{T}_2 \cup \{\langle {}^{\triangleright}t, {}^{\triangleleft}t \rangle \mid \langle \langle {}^{\triangleright}t, \emptyset \rangle, \langle \{\iota_2\}, {}^{\triangleleft}t \rangle \rangle \in \mathscr{T}_1 \times \mathscr{T}_2\}$. We may present this transition system with inference rules as follows:

$$\frac{\mathscr{A}_1 \vdash t : S \to T, \quad T \neq \emptyset}{\mathscr{A}_1 \cdot \mathscr{A}_2 \vdash t : S \to T} \qquad \frac{\mathscr{A}_2 \vdash t : S \to T}{\mathscr{A}_1 \cdot \mathscr{A}_2 \vdash t : S \to T}$$

$$\frac{\mathscr{A}_1 \vdash \langle {}^{\triangleright}t, \emptyset \rangle : S \to \emptyset, \quad \mathscr{A}_2 \vdash \langle \{\iota_2\}, {}^{\triangleleft}t \rangle : \{\iota_2\} \to T}{\mathscr{A}_1 \cdot \mathscr{A}_2 \vdash \langle {}^{\triangleright}t, {}^{\triangleleft}t \rangle : S \to T}$$

Remark. The inference system above is not entirely faithful to the above definition, in the sense that the last rule should be:

$$\frac{\mathscr{A}_1 \vdash \langle {}^{\triangleright}t, \emptyset \rangle : S \to S', \quad \mathscr{A}_2 \vdash \langle \{\iota_2\}, {}^{\triangleleft}t \rangle : \{\iota_2\} \to T}{\mathscr{A}_1 \cdot \mathscr{A}_2 \vdash \langle {}^{\triangleright}t, {}^{\triangleleft}t \rangle : S \to S' \cup T}.$$

However, the application of this rule in the case where $S' \neq \emptyset$ cannot yield an accepting run without $\mathscr{A}_1$ violating Constraint 2. Indeed if there was an accepting run in $\mathscr{A}_1 \cdot \mathscr{A}_2$ using this rule, this run would need at least another occurrence of the same rule to consume the remaining places in P_1, thus allowing one to build an accepting run in $\mathscr{A}_1$ with several final transitions. This contradicts Lemma 3.10, thus violating Constraint 2. Phenomena of the same nature occur in the construction of $\mathscr{A}_1^+$ and $\mathscr{A}_1 \cap \mathscr{A}_2$, thus explaining the slight discrepancies between the definitions of the automata and the inference systems given. This inference system should thus be construed as a way to get intuitions rather than a formal definition.

Lemma 3.12. $\mathscr{G}(\mathscr{A}_1 \cdot \mathscr{A}_2) = \mathscr{G}(\mathscr{A}_1) \cdot \mathscr{G}(\mathscr{A}_2)$. ■

Proof. Let $G \in \mathscr{G}(\mathscr{A}_1) \cdot \mathscr{G}(\mathscr{A}_2)$. There are two accepting runs $R_i = \langle \{\iota_i\}, t_0^i; \cdots; t_{n_i}^i, \emptyset \rangle$ (with $i \in \{1, 2\}$) such that $G \equiv \mathscr{G}(R_1) \cdot \mathscr{G}(R_2)$. Let us name some intermediary states: we call S_1 and S_2 the states such that:

$$\mathscr{A}_1 \vdash t_0^1; \dots; t_{n_1-1}^1 : \{\iota_1\} \to S_1; \qquad \mathscr{A}_1 \vdash t_{n_1}^1 : S_1 \to \emptyset;$$

$$\mathscr{A}_2 \vdash t_0^2 : \{\iota_2\} \to S_2; \qquad \mathscr{A}_2 \vdash t_1^2; \dots; t_{n_2}^2 : S_2 \to \emptyset.$$

Because of Lemma 3.10, we know that in R_1, only $t^1_{n_1}$ is a final transition. Using the inference rules above, we obtain:

$$\mathscr{A}_1 \cdot \mathscr{A}_2 \vdash t^1_0; \ldots; t^1_{n_1-1} : \{\iota_1\} \to S_1; \qquad \mathscr{A}_1 \cdot \mathscr{A}_2 \vdash t^2_1; \ldots; t^2_{n_2} : S_2 \to \emptyset;$$

$$\mathscr{A}_1 \cdot \mathscr{A}_2 \vdash \left\langle {}^{\triangleright}t^1_{n_1}, {}^{\triangleleft}t^2_0 \right\rangle : S_1 \to S_2.$$

Thus we obtain $R = \left\langle \{\iota_1\}, t^1_0; \ldots; t^1_{n_1-1}; \left\langle {}^{\triangleright}t^1_{n_1}, {}^{\triangleleft}t^2_0 \right\rangle; t^2_1; \ldots; t^2_{n_2}, \emptyset \right\rangle$ which is accepting in $\mathscr{A}_1 \cdot \mathscr{A}_2$ and satisfies $\mathscr{G}(R) \equiv \mathscr{G}(R_1) \cdot \mathscr{G}(R_2) \equiv G$.

For the other direction, let $G \in \mathscr{G}(\mathscr{A}_1 \cdot \mathscr{A}_2)$ be a graph, and $R = \langle \{\iota_1\}, t_0; \ldots; t_n, \emptyset \rangle$ be the corresponding accepting run. We can extract from this run two runs R_1 and R_2 respectively in $\mathscr{A}_1$ and $\mathscr{A}_2$ as follows:

$$R^0_1 := \varepsilon, \qquad R^{k+1}_1 := \begin{cases} R^k_1; t_k & (\text{if } t_k \in \mathscr{T}_1) \\ R^k_1; \left\langle {}^{\triangleright}t_k, \emptyset \right\rangle & \left(\text{if } \left\langle \left\langle {}^{\triangleright}t_k, \emptyset \right\rangle, \left\langle \{\iota_2\}, {}^{\triangleleft}t_k \right\rangle \right\rangle \in \mathscr{T}_1 \times \mathscr{T}_2\right) \\ R^k_1 & (\text{otherwise}) \end{cases},$$

$$R^0_2 := \varepsilon, \qquad R^{k+1}_2 := \begin{cases} R^k_2; t_k & (\text{if } t_k \in \mathscr{T}_2) \\ R^k_2; \left\langle \{\iota_2\}, {}^{\triangleleft}t_k \right\rangle & \left(\text{if } \left\langle \left\langle {}^{\triangleright}t_k, \emptyset \right\rangle, \left\langle \{\iota_2\}, {}^{\triangleleft}t_k \right\rangle \right\rangle \in \mathscr{T}_1 \times \mathscr{T}_2\right) \\ R^k_2 & (\text{otherwise}) \end{cases},$$

$$R_1 := R^{n+1}_1, \qquad R_2 := R^{n+1}_2.$$

Using these definitions, we may prove that $\forall k \leqslant n$, $\mathscr{A}_1 \cdot \mathscr{A}_2 \vdash t_0; \ldots; t_k : \{\iota_1\} \to S$ entails $\mathscr{A}_1 \vdash R^{k+1}_1 : \{\iota_1\} \to (S \setminus P_2)$. This means R_1 is an accepting run in $\mathscr{A}_1$. By Lemma 3.10 this entails there is a single final transition in R_1, hence a single transition t_i in R such that $\left\langle \left\langle {}^{\triangleright}t_i, \emptyset \right\rangle, \left\langle \{\iota_2\}, {}^{\triangleleft}t_i \right\rangle \in \mathscr{T}_1 \times \mathscr{T}_2 \right\rangle$. Again by Lemma 3.10 we know that t_i marks the end of R_1, and clearly R_2 cannot begin before t_i has been fired (no place in P_2 can appear before that). From this we can deduce that R has almost the shape $R_1; R_2$ (but with the two transitions in the middle merged into t_i). We can also check that $\forall k \geqslant i$, $\mathscr{A}_1 \cdot \mathscr{A}_2 \vdash t_0; \ldots; t_k : \{\iota_1\} \to S$ entails $\mathscr{A}_2 \vdash R^{k+1}_2 : \{\iota_2\} \to (S \setminus P_1)$, thus proving that R_2 is an accepting run in $\mathscr{A}_2$. Finally we get that $\mathscr{G}(R) \equiv \mathscr{G}(R_1) \cdot \mathscr{G}(R_2) \in \mathscr{G}(\mathscr{A}_1) \cdot \mathscr{G}(\mathscr{A}_2)$. □

Then iterating the automaton $\mathscr{A}_1$ can be done by looping the previous construction on itself: we keep the places and transitions of the automaton, but simply add a transition $\langle {}^{\triangleright}t, {}^{\triangleleft}t \rangle$ for every pair of an initial transition $\langle \{\iota_1\}, {}^{\triangleleft}t \rangle$ and a final transition $\langle {}^{\triangleright}t, \emptyset \rangle$ in $\mathscr{T}_1$.

$$\mathscr{A}_1^+ := \langle P_1, \mathscr{T}_1 \cup \{\langle {}^{\triangleright}t, {}^{\triangleleft}t \rangle \mid \langle \langle {}^{\triangleright}t, \emptyset \rangle, \langle \{\iota_1\}, {}^{\triangleleft}t \rangle \rangle \in \mathscr{T}_1 \times \mathscr{T}_1\}, \iota_1 \rangle.$$

As an inference system, we get:

$$\frac{\mathscr{A}_1 \vdash t : S \to T}{\mathscr{A}_1^+ \vdash t : S \to T} \qquad \frac{\mathscr{A}_1 \vdash \langle {}^{\triangleright}t, \emptyset \rangle : S \to \emptyset, \quad \mathscr{A}_1 \vdash \langle \{\iota_1\}, {}^{\triangleleft}t \rangle : \{\iota_1\} \to T}{\mathscr{A}_1^+ \vdash t : S \to T}$$

Lemma 3.13. $\mathscr{G}\left(\mathscr{A}_1^+\right) = \mathscr{G}(\mathscr{A}_1)^+$. ■

Proof. The proof follows the same scheme as the previous one. □

Parallelisation of automata. We still have $\mathscr{A}_1$ and $\mathscr{A}_2$ with disjoint sets of places. We can get an automaton whose traces are obtained by the parallel product of traces of $\mathscr{A}_1$ and $\mathscr{A}_2$ by merging the initial transitions of the two automata, and then merging their final transitions. This yields the following automaton:

$$\mathscr{A}_1 \cap \mathscr{A}_2 := \left\langle P_1 \cup P_2 \cup \{\iota\}, (\mathscr{T}_1 \setminus \mathcal{P}(P_1) \times \{\emptyset\}) \cup (\mathscr{T}_1 \setminus \mathcal{P}(P_2) \times \{\emptyset\}) \cup \mathscr{T}^i \cup \mathscr{T}_f, \iota \right\rangle,$$

where $\mathscr{T}^i := \{\langle \{\iota\}, {}^{\triangleleft}t_1 \cup {}^{\triangleleft}t_2 \rangle \mid \langle \langle \{\iota_1\}, {}^{\triangleleft}t_1 \rangle, \langle \{\iota_2\}, {}^{\triangleleft}t_2 \rangle \rangle \in \mathscr{T}_1 \times \mathscr{T}_2\}$
and $\mathscr{T}^f := \{\langle {}^{\triangleright}t_1 \cup {}^{\triangleright}t_2, \emptyset \rangle \mid \langle \langle {}^{\triangleright}t_1, \emptyset \rangle, \langle {}^{\triangleright}t_2, \emptyset \rangle \rangle \in \mathscr{T}_1 \times \mathscr{T}_2\}$.

Notice that because P_1 and P_2 are of empty intersection, the set of states of this automaton (i.e. $\mathcal{P}(P_1 \cup P_2 \cup \{\iota\})$) is isomorphic to $\mathcal{P}(P_1) \times \mathcal{P}(P_2) \times \mathcal{P}(\{\iota\})$. For clarity, we use the later notation. Presented as an inference system, the definition above stands for:

$$\frac{\mathscr{A}_1 \vdash t : S \to T, \quad {}^{\triangleleft}t \neq \emptyset}{\mathscr{A}_1 \cap \mathscr{A}_2 \vdash t : \langle S, X, \emptyset\rangle \to \langle T, X, \emptyset\rangle} \qquad \frac{\mathscr{A}_2 \vdash t : S \to T, \quad {}^{\triangleleft}t \neq \emptyset}{\mathscr{A}_1 \cap \mathscr{A}_2 \vdash t : \langle X, S, \emptyset\rangle \to \langle X, T, \emptyset\rangle}$$

$$\frac{\mathscr{A}_1 \vdash \langle \{\iota_1\}, {}^{\triangleleft}t \setminus (\Sigma \times P_2)\rangle : \{\iota_1\} \to S_1, \quad \mathscr{A}_2 \vdash \langle \{\iota_2\}, {}^{\triangleleft}t \setminus (\Sigma \times P_1)\rangle : \{\iota_2\} \to S_2}{\mathscr{A}_1 \cap \mathscr{A}_2 \vdash t : \langle \emptyset, \emptyset, \{\iota\}\rangle \to \langle S_1, S_2, \emptyset\rangle}$$

$$\frac{\mathscr{A}_1 \vdash \langle {}^{\triangleright}t \setminus P_2, \emptyset\rangle : S_1 \to \emptyset, \quad \mathscr{A}_2 \vdash \langle {}^{\triangleright}t \setminus P_1, \emptyset\rangle : S_2 \to \emptyset, \quad {}^{\triangleleft}t = \emptyset}{\mathscr{A}_1 \cap \mathscr{A}_2 \vdash t : \langle S_1, S_2, \emptyset\rangle \to \langle \emptyset, \emptyset, \emptyset\rangle}$$

Lemma 3.14. $\mathscr{G}(\mathscr{A}_1 \cap \mathscr{A}_2) = \mathscr{G}(\mathscr{A}_1) \cap \mathscr{G}(\mathscr{A}_2)$. ■

Proof. Let $G \in \mathscr{G}(\mathscr{A}_1) \cap \mathscr{G}(\mathscr{A}_2)$. There are two accepting runs in $\mathscr{A}_1$ and $\mathscr{A}_2$, that we call $R_i = \langle \{\iota_i\}, t_0^i; \ldots; t_{n_i}^i, \emptyset\rangle$ (with $i \in \{1,2\}$), such that $G \equiv \mathscr{G}(R_1) \cap \mathscr{G}(R_2)$. We build the following run:

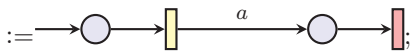
$$R := \left\langle \{\iota\}, \left\langle \{\iota\}, {}^{\triangleleft}t_0^1 \cup {}^{\triangleleft}t_0^2 \right\rangle; t_1^1; \ldots; t_{n_1-1}^1; t_1^2; \ldots; t_{n_2-1}^2; \left\langle {}^{\triangleright}t_{n_1}^1 \cup {}^{\triangleright}t_{n_2}^2, \emptyset \right\rangle, \emptyset \right\rangle.$$

It is a simple exercise to check that indeed R is an accepting run in $\mathscr{A}_1 \cap \mathscr{A}_2$, and that its trace does satisfy $\mathscr{G}(R) \equiv G$.

For the other direction, the presentation as an inference system simplifies the reasoning: quite clearly, by projecting an accepting run in $\mathscr{A}_1 \cap \mathscr{A}_2$ on the first (respectively second) component, we get an accepting run in $\mathscr{A}_1$ (resp. $\mathscr{A}_2$). From that remark, one can deduce that the parallel product of the traces of these two projected runs is isomorphic to the trace of the whole run. □

Conclusion. We now formally define the function $\mathscr{A}(\_)$:

Definition 3.15 ($\mathscr{A}(e)$)**.**
We define $\mathscr{A}(e)$ by induction on e:

$\mathscr{A}(a) :=$ 

; $\quad \mathscr{A}(0) :=$  ;

$\mathscr{A}(e \cdot f) := \mathscr{A}(e) \cdot \mathscr{A}(f)$; $\quad \mathscr{A}(e + f) := \mathscr{A}(e) \cup \mathscr{A}(f)$;

$\mathscr{A}(e \cap f) := \mathscr{A}(e) \cap \mathscr{A}(f)$; $\quad \mathscr{A}\left(e^+\right) := \mathscr{A}(e)^+$. ∗

It is then straightforward to check that the automaton produced by this function recognises the intended language, using Lemmas 3.11 to 3.14.

Lemma 3.16 (Correctness of $\mathscr{A}(\_)$)**.** $\forall e \in GReg\langle\Sigma\rangle, \mathscr{G}(\mathscr{A}(e)) \equiv \mathscr{G}(e)$. ■

This construction allows us to state the first half of a Kleene theorem:

Theorem 3.17. *Regular sets of graphs are recognisable.* ■

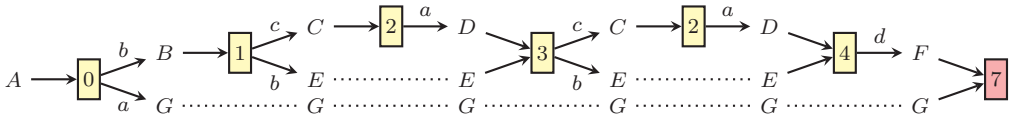
Proof. If S is a regular set of graphs, then there is an expression $e \in \mathrm{GReg}\langle\Sigma\rangle$ such that $S \equiv \mathscr{G}(e)$. By Lemma 3.16, $\mathscr{G}(\mathscr{A}(e)) \equiv \mathscr{G}(e) \equiv S$, thus proving S is recognisable. □

Naturally, we would like to establish the converse as well, to fully equate these two classes of graph languages. This will be the focus of Section 3.4.

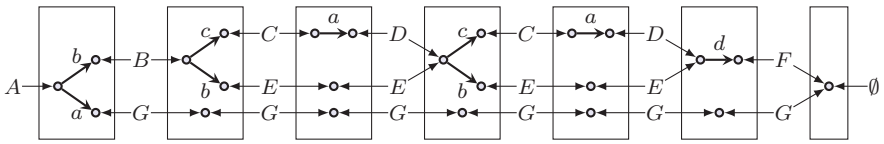
Remark. If e is an expression without intersection, it can be shown that the transitions in $\mathscr{A}(e)$ are all of the form $\langle \{p\}, \{\langle x, q\rangle\}\rangle$, with only one input, one output and a label in Σ. As a consequence, the accessible configurations are singletons, and the resulting Petri automaton has the structure of a non-deterministic finite-state automaton (NFA). Actually, in that case, the construction we described above is just a variation on Thompson's construction [60], with inlined epsilon transition elimination.

3.3 Boxes

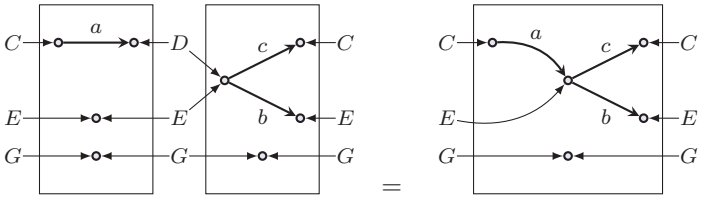
To get a full Kleene theorem, a key ingredient is the ability to represent the labelling of a fragment of execution of some automaton. As Petri automata have crucially a concurrent component, words will not do the trick. For that reason, we introduce boxes, which represent "slices" of TTSP graphs, with distinguished inputs and outputs. Recall the following run, that we used as an example of run from the automaton displayed in Figure 3.4:



We will abstract each transition by a box, and the run itself by the sequential composition of those boxes:



These boxes allow us to represent any subrun as a single box. For instance, the subrun firing transitions 2 and 3 in sequence, and looping from state $\{C, E, G\}$ back to itself can be represented as the box:



In this section, we describe the formal definition of these boxes, and we study some of their properties.

3.3.1 Categories of boxes

We fix a finite set P of ports, which will be instantiated in the next section with the set of places of some Petri automaton.

Definition 3.18 (Box)**.**
Let $S, S' \subseteq P$ be two sets of ports. A *box* labelled over Σ from S to S' is a triple $\langle \overrightarrow{\mathfrak{p}}, G, \overleftarrow{\mathfrak{p}} \rangle$ where G is a DAG labelled with Σ, $\overrightarrow{\mathfrak{p}}$ is a map from S to the vertices of G, and $\overleftarrow{\mathfrak{p}}$ is a bijective map from S' to $\max G$.

The set of boxes labelled by Σ is written $\mathcal{B}_\Sigma$, and the boxes from S to S' are denoted by $\mathcal{B}_\Sigma(S, S')$, or simply $\mathcal{B}(S, S')$ if the set of labels is clear from the context. ∗

As for graphs, we consider boxes up to renaming of internal nodes. We use graphical representations for boxes, as illustrated in Figure 3.7. Inside the rectangle is the DAG, with on the left the input ports and on the right the output ports. The maps $\overrightarrow{\mathfrak{p}}$ and $\overleftarrow{\mathfrak{p}}$ are represented by the arrows going from the ports to vertices inside the rectangle.

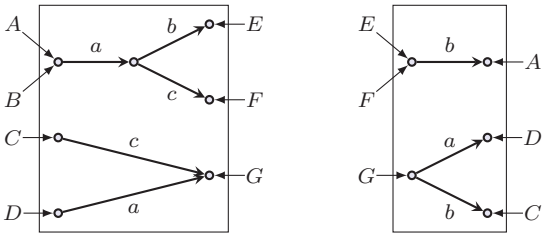


Figure 3.7: Examples of boxes

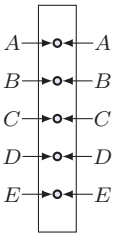


Figure 3.8: id_σ.

Definition 3.19 (Identity box)**.**
If $S = \{p_1, \ldots, p_n\} \subseteq P$ is a set of ports, the *identity box* on S is defined as $\mathrm{id}_S := \langle [p_i \mapsto i], \langle \{1, \ldots, n\}, \emptyset \rangle, [p_i \mapsto i] \rangle$. *

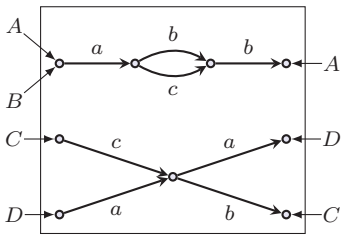
The box $\mathrm{id}_{\{A,B,C,D,E\}}$ is represented in Figure 3.8.

One of the most interesting features of boxes is their ability to compose. Intuitively, if the set S of output ports of β_1 is equal to the set of input ports of β_2, we may compose them by putting the graph of β_1 to the left of the graph of β_2, and for every port $p \in S$, we identify the node $\overleftarrow{\mathfrak{p}_1}(p)$ with the node $\overrightarrow{\mathfrak{p}_2}(p)$.

Definition 3.20 (Composition of boxes)**.**
Let $S_1, S_2, S_3 \subseteq P$ and for $i \in \{1, 2\}$, let β_i be a box from S_i to S_{i+1}, with $\beta_i = \langle \overrightarrow{\mathfrak{p}_i}, \langle V_i, E_i \rangle, \overleftarrow{\mathfrak{p}_i} \rangle$, such that $V_1 \cap V_2 = \emptyset$. The *composition* of β_1 and β_2, written $\beta_1 \odot \beta_2$ may be defined as $\langle \overrightarrow{\mathfrak{p}}, \langle V_1' \cup V_2, E_1' \cup E \cup E_2 \rangle, \overleftarrow{\mathfrak{p}} \rangle$ with:

- $V_1' := V_1 \setminus \overleftarrow{\mathfrak{p}_1}(S_2)$, and $E_1' := E_1 \cap (V_1' \times \Sigma \times V_1')$;
- $E := \left\{ \left\langle x, a, \overrightarrow{\mathfrak{p}_2}(p) \right\rangle \;\middle|\; \langle x, a, y \rangle \in E_1,\ y = \overleftarrow{\mathfrak{p}_1}(p) \right\}$;
- $\overleftarrow{\mathfrak{p}} := \overleftarrow{\mathfrak{p}_2}$ and $\overrightarrow{\mathfrak{p}}(p) := \begin{cases} \overrightarrow{\mathfrak{p}_2}(q), & \text{if } \overrightarrow{\mathfrak{p}_1}(p) = \overleftarrow{\mathfrak{p}_1}(q), \\ \overrightarrow{\mathfrak{p}_1}(p), & \text{otherwise.} \end{cases}$ *

For instance the two boxes in Figure 3.7 may be composed, thus yielding the following box:



It can be used to define a category of sets of ports and boxes:

Lemma 3.21. *One may form a category $\mathcal{B}ox_P^\Sigma$ whose objects are subsets of P, and whose morphisms between S and S' are the boxes from S to S', i.e. $\mathcal{B}_\Sigma(S, S')$.* ■

Proof. To give a high level proof of the associativity of box composition, notice that the computation of $\beta \odot \gamma$ may be split in two steps:

1. compute an intermediate box $\beta \xrightarrow{\varepsilon} \gamma$ (with ε being a fresh label), build by keeping the input port map of β, the output map of γ, the disjoint union of their vertices and edges, and simply adding edges $x \xrightarrow{\varepsilon} y$ whenever $\langle x, y \rangle \in \max \beta \times \gamma$ such that $\overleftarrow{\mathfrak{p}_\beta}(x) = \overrightarrow{\mathfrak{p}_\gamma}(y)$.

2. collapse all edges labelled by ε by identifying their source and target vertices. The effect of this will be the destruction of all the nodes from $\max\beta$, and the redirection of their incoming arrows to the corresponding vertices in γ.

Suppose we have three boxes β, γ, δ with the appropriate input and output sets of ports (the input ports of γ should be exactly the output ports of β...). We may now describe the two ways in which to compose them by the following diagram:

$$\begin{array}{ccccc}
\beta,\gamma,\delta & \longrightarrow & \beta,\gamma \xrightarrow{\varepsilon_2} \delta & \longrightarrow & \beta,\gamma\odot\delta \\
\downarrow & (1) & \downarrow & (2) & \downarrow \\
\beta \xrightarrow{\varepsilon_1} \gamma,\delta & \longrightarrow & \beta \xrightarrow{\varepsilon_1} \gamma \xrightarrow{\varepsilon_2} \delta & \longrightarrow & \beta \xrightarrow{\varepsilon_1} \gamma\odot\delta \\
\downarrow & (3) & \downarrow & (4) & \downarrow \\
\beta\odot\gamma,\delta & \longrightarrow & \beta\odot\gamma \xrightarrow{\varepsilon_2} \delta & \longrightarrow & \beta\odot\gamma\odot\delta
\end{array}$$

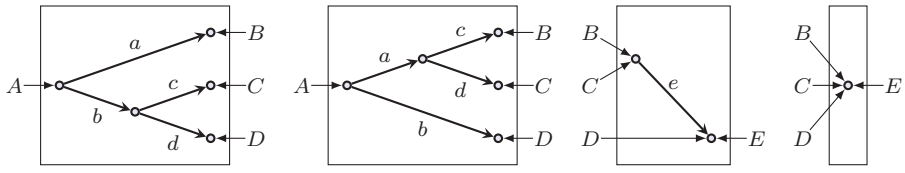
The commutation of square (1) is clear: the box $\beta \xrightarrow{\varepsilon_1} \gamma \xrightarrow{\varepsilon_2} \delta$ could even be described in one step (just put edges with ε_1 between $\max\beta$ and γ and with ε_2 between $\max\gamma$ and δ.

Squares (2) and (3) commute as well. For instance for square (3), notice that collapsing edges ε_1 in $\beta \xrightarrow{\varepsilon_1} \gamma$ doesn't affect the set $\max\beta \xrightarrow{\varepsilon_1} \gamma = \max\gamma$. Hence this has no bearing on the computation of $\beta\odot\gamma \xrightarrow{\varepsilon_2} \delta$.

Finally, for square (4), an additional step could be added: replacing both labels ε_1 and ε_2 by a common label ε, before collapsing the resulting graph along ε-edges. This clearly produces the same result. The point is then to show that the collapsing operation is confluent. This is true, because one could compute the normal form from the beginning, as the set of equivalence classes of the smallest equivalence relation on vertices containing all pairs x, y of vertices linked by an ε-edge.

The fact that for any box $\beta = \left\langle \overrightarrow{\mathfrak{p}}, \langle V, E\rangle, \overleftarrow{\mathfrak{p}} \right\rangle \in \mathcal{B}(A,B)$, we have $\mathrm{id}_A \odot \beta = \beta$ is straightforward from the definitions. The vertices of the composite box are exactly those of β, because the image of A through the output map of id_A is the set of vertices of id_A itself. As id_A has no edge, the edges of $\mathrm{id}_A \odot \beta$ are again simply those of β. By definition the output map is that of β, but so is the input map, because for every port $p_i \in A$, the images of p_i via the input and output maps of id_A are both equal to i. The fact that $\beta \odot \mathrm{id}_B$ is also equal to β follows from a similar argument. □

We actually need to enforce a stronger typing discipline on boxes. Intuitively, this stems from our desire to use these boxes to represent fragments of series-parallel graphs. However in the current setting, we may very well have two boxes that are indeed fragments of series parallel graphs, which are allowed to be composed, but whose composition cannot be a part of a SP-graph. For instance consider the four boxes below, called β_1 through β_4:



According to their interfaces, β_1 and β_2 can both compose with β_3 and β_4. Indeed $\beta_1 \odot \beta_4$, $\beta_2 \odot \beta_3$ and $\beta_2 \odot \beta_4$ all yield boxes containing TTSP graphs. However the composition $\beta_1 \odot \beta_3$ produces the box shown in Figure 3.9. The graph contained in this box is not TTSP, and is in fact the forbidden subgraph of the class of TTSP graphs [61]. To prevent this situation statically, we introduce typing. The types we use are top-down trees with leaves labelled with ports:

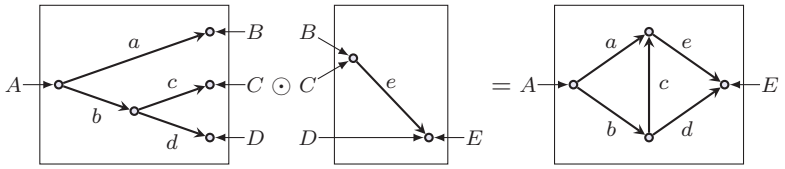


Figure 3.9: An example of "bad" composition

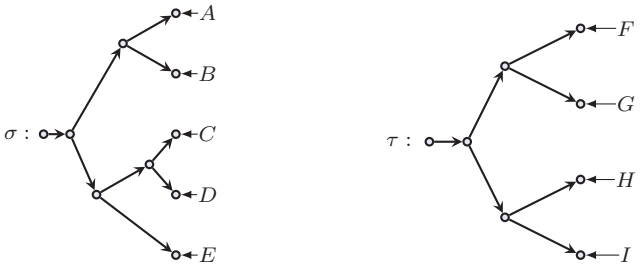


Figure 3.10: Example of types

Definition 3.22 (Type)**.**
A *type* over $S \subseteq P$ is a triple $\tau = \langle V, E, \lambda \rangle$ such that $\langle V, E \rangle$ is a proper unlabelled top-down tree, and λ is a bijective function from S to $\max \langle V, E \rangle$. The set of types over subsets of P is written $\mathbb{T}_P$. ∗

As before, types are considered up to bijective renaming of internal vertices. It is then a simple observation to notice that $\mathbb{T}_P$ is finite (recall that P was assumed to be finite as well). We present two examples of such types in Figure 3.10.

We may forget about the label information in a box: the a-erasing of a box β is the box $\lfloor \beta \rfloor_a$ where all labels are replaced by an arbitrary letter $a \in \Sigma$. When the labels are not relevant (or, in the next section when we label edges with expressions rather than letters), we can define SP-reductions of boxes. Given a box $\beta = \left\langle \overrightarrow{\mathfrak{p}}, G, \overleftarrow{\mathfrak{p}} \right\rangle$, if $G \to_{SP} G'$, and if no vertex in the image of $\overrightarrow{\mathfrak{p}}$ was erased in the reduction, we write $\beta \to_{SP} \left\langle \overrightarrow{\mathfrak{p}}, G', \overleftarrow{\mathfrak{p}} \right\rangle$. Composition commutes with SP-reductions: if $\beta \to_{SP} \beta'$, then $\beta \odot \gamma \to_{SP} \beta' \odot \gamma$ and $\gamma \odot \beta \to_{SP} \gamma \odot \beta'$.

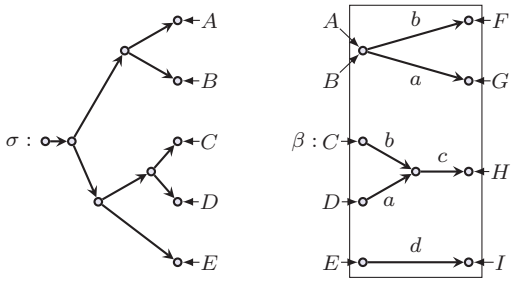
A type $\tau = \langle V, E, \lambda \rangle$ over S may be seen as a box from $\{\iota\}$ to S labelled with $\Sigma \neq \emptyset$: if r is the root of τ and $a \in \Sigma$, we can build the box $\boxed{\tau}_a$ from $\{\iota\}$ to S as $\langle [\iota \mapsto r], \langle V, E' \rangle, \lambda \rangle$, with $E' = \{\langle x, a, y \rangle \mid \langle x, y \rangle \in E\}$. It is quite clear that for any type τ, $\boxed{\tau}_a = \left\lfloor \boxed{\tau}_a \right\rfloor_a$, and that for any two boxes $\beta \in \mathcal{B}(S_1, S_2)$ and $\gamma \in \mathcal{B}(S_2, S_3)$, we have $\lfloor \beta \odot \gamma \rfloor_a = \lfloor \beta \rfloor_a \odot \lfloor \gamma \rfloor_a$.

We may now use these to type boxes, in the following way.

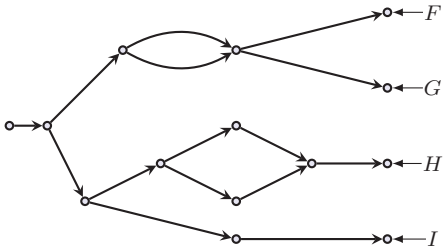
Definition 3.23 (Typed boxes)**.**
Let $\beta \in \mathcal{B}(S, S')$ be a box, σ and τ be types respectively over S and S'. *β has the type* $\sigma \to \tau$ if $\left\lfloor \boxed{\sigma}_a \odot \beta \right\rfloor_a \to^\star_{SP} \boxed{\tau}_a$. We write $\mathfrak{B}_\Sigma(\sigma, \tau)$ for the set of boxes over Σ of type $\sigma \to \tau$, and $\mathfrak{B}_\Sigma$ for the set of typed boxes over Σ. ∗

Remark. Notice that the type of a box is not unique: a single box may have multiple types. However, given an input type σ and a box β, there is at most one output type τ such that $\beta \in \mathfrak{B}(\sigma, \tau)$.

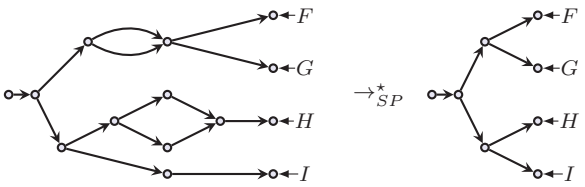
Example 3.24.
Consider the following type σ and box β:



First, we glue them together, and erase the labels. This yields the graph:



Then, we apply as many reductions steps as much as possible:



The last graph is the box corresponding to a tree τ. Hence we can state that β has the type $\sigma \to \tau$. •

The composition of typed boxes is a typed box and the identity box can be typed.

Lemma 3.25. *$\mathfrak{B}(\sigma,\tau)\odot\mathfrak{B}(\tau,\theta) \subseteq \mathfrak{B}(\sigma,\theta)$ and if σ is a type over S, then $id_S \in \mathfrak{B}(\sigma,\sigma)$.* ■

Proof. Let $\beta,\gamma \in \mathfrak{B}(\sigma,\tau) \times \mathfrak{B}(\tau,\theta)$, we show that $\beta \odot \gamma \in \mathfrak{B}(\sigma,\theta)$:

$\left\lfloor \boxed{\sigma}_a \odot (\beta \odot \gamma) \right\rfloor_a = \left\lfloor \boxed{\sigma}_a \odot \beta \right\rfloor_a \odot \lfloor\gamma\rfloor_a \to^\star_{SP} \boxed{\tau}_a \odot \lfloor\gamma\rfloor_a = \left\lfloor \boxed{\tau}_a \odot \gamma \right\rfloor_a \to^\star_{SP} \boxed{\theta}_a.$

$\left\lfloor \boxed{\sigma}_a \odot \mathrm{id}_S \right\rfloor_a = \left\lfloor \boxed{\sigma}_a \right\rfloor_a = \boxed{\sigma}_a.$ □

For this reason we write $\mathrm{id}_\sigma := \mathrm{id}_S$.

Corollary 3.26. *There is a category $\mathcal{TB}ox^\Sigma_P$ of typed boxes, with $\mathbb{T}_P$ as the set of objects, and $\mathfrak{B}_\Sigma(\tau,\sigma)$ as the set of morphisms between τ and σ.* ■

Remark. It is clear that whenever we have a category $\mathcal{C}$, we can form a typed Kleene algebra [38, 48] $\mathcal{K}$ whose atomic types are the objects of the category, and where the set of elements of type $a \to b$ is the set $\mathcal{H}om_\mathcal{C}(a,b)$ of homomorphisms from a to b. The product in this algebra is the pointwise composition of homomorphisms, and the sum is the union. This means that $\mathcal{P}(\mathcal{B})$ and $\mathcal{P}(\mathfrak{B})$ are typed Kleene algebras.

3.3.2 Templates

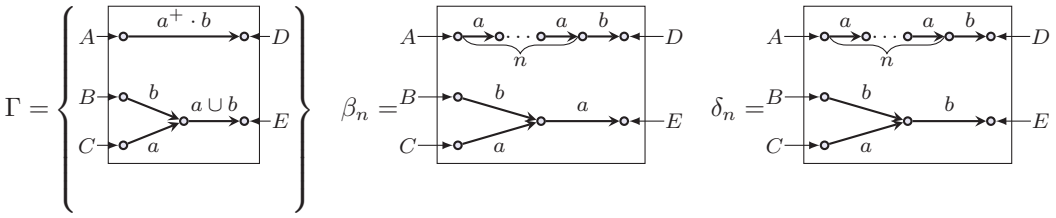
Another important notion we need for the second direction of our Kleene theorem is that of template. In the proof of the classical Kleene theorem, one moves from automata to generalised automata, which are labelled with regular expressions rather than with letters. This allows for the label of a single transition to represent many paths in the original automaton. Finite templates will serve this function in our proof. A finite template is a finite set of boxes sharing the same input an output ports, and labelled with graph expressions.

Definition 3.27 ((Finite) Template)**.**
Let S and S' be sets of of ports. A *template* (respectively *finite template*) from S to S' is a set (resp. finite set) $\Gamma \subseteq \mathcal{B}_{\mathrm{GReg}\langle\Sigma\rangle}(S, S')$ of boxes from S to S' labelled with expressions. If furthermore σ is a type over S, τ is a type over S' and $\Gamma \subseteq \mathfrak{B}_{\mathrm{GReg}\langle\Sigma\rangle}(\sigma, \tau)$, we say that Γ has type $\sigma \to \tau$. We write $\mathcal{BT}(S, S')$ for the set of finite templates from S to S', and $\mathfrak{BT}(\sigma, \tau)$ for finite templates of type $\sigma \to \tau$. ∗

From such sets of boxes, one may extract "standard" boxes, as follows:

Definition 3.28 (Boxes generated by a template)**.**
Let $\Gamma \in \mathcal{BT}(S, S')$ be a template from S to S'. Then $\beta \in \mathcal{B}_\Sigma(S, S')$ is *generated* by Γ if it can be obtained from a box $\left\langle \overrightarrow{\mathfrak{p}}, G, \overleftarrow{\mathfrak{p}} \right\rangle \in \Gamma$ by replacing each edge $x \xrightarrow{e} y$ by a TTSP graph $G' \in \mathscr{G}(e)$ with input vertex x and output vertex y. We write $(\!|\Gamma|\!)$ for the set of boxes generated by Γ. ∗

For instance in the example below the template Γ can generate all boxes of the shapes β_n and δ_n, for $n > 0$:



Remark. If Γ has the type $\sigma \to \tau$, then $(\!|\Gamma|\!) \subseteq \mathfrak{B}_\Sigma(\sigma, \tau)$.

As per Lemma 3.21, Corollary 3.26 and the subsequent remark, the set of templates and the set of typed templates form typed Kleene algebras. Hence, we define the regular operations on templates.

Definition 3.29 (Regular operations on templates)**.**

- $1_S := \{\mathrm{id}_S\}$ and $0 := \emptyset$.
- Let $\Gamma, \Delta \subseteq \mathcal{B}(S, S') \times \mathcal{B}(S', S'')$, then $\Gamma \cdot \Delta := \Gamma \odot \Delta$.
- Let $\Gamma, \Delta \subseteq \mathcal{B}(S, S')$, then $\Gamma + \Delta := \Gamma \cup \Delta$.
- Let $\Gamma \subseteq \mathcal{B}(S, S)$, then $\Gamma^\star := \bigcup_{n \in \mathbb{N}} \underbrace{\Gamma \odot \cdots \odot \Gamma}_{n}$. ∗

These operations are compatible with types. One can easily check that 1_σ is the unit of the product, 0 is the unit of the sum, and that these definitions are compatible with the sets of generated boxes:

$$(\!|1_\sigma|\!) = \{\mathrm{id}_\sigma\} \qquad (\!|0|\!) = \emptyset \qquad (\!|\Gamma \cdot \Delta|\!) = (\!|\Gamma|\!) \odot (\!|\Delta|\!) \qquad (\!|\Gamma + \Delta|\!) = (\!|\Gamma|\!) \cup (\!|\Delta|\!). \qquad (\!|\Gamma^\star|\!) = (\!|\Gamma|\!)^\star.$$

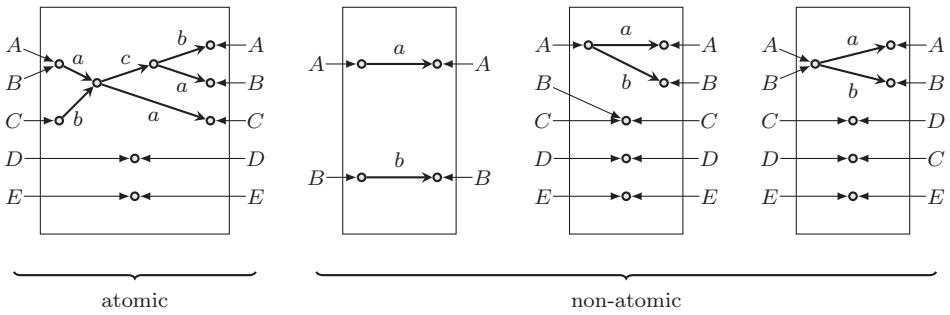
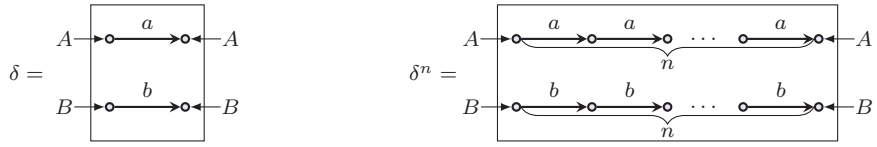


Figure 3.11: Atomic and non-atomic boxes

Furthermore, if Γ_1, Γ_2 and Γ_3 are finite templates of appropriate type, then $\Gamma_1 + \Gamma_2$ and $\Gamma_1 \cdot \Gamma_2$ are also finite typed templates.

However the Kleene star of a set of typed boxes yields an infinite set of boxes. Worse, the language it generates cannot always be represented as a finite template. For instance, the star of the box δ below yields all boxes δ^n, for $n \in \mathbb{N}$:



This set of boxes cannot be represented by a finite number of boxes, as the two branches have no way to synchronise. (This means that we cannot ensure there will be exactly the same number of iterations on both branches.) Fortunately, it is possible to define this operation for a rich enough class of templates, namely for *atomic* templates.

3.3.3 Atomic boxes and templates

Definition 3.30 (support)**.**
Let $\beta \in \mathcal{B}(S,S)$, the *support* of β is $\mathit{support}(\beta) := \{p \mid \overrightarrow{\mathfrak{p}}(p) \neq \overleftarrow{\mathfrak{p}}(p)\}$. The support of a template $\Gamma \in \mathcal{BT}(S,S)$ is then defined as $\bigcup_{\beta\in\Gamma} \mathit{support}(\beta)$. ∗

Intuitively, the support constitutes the irreflexive part of a box. In particular, $\mathit{support}(\mathrm{id}_\sigma)$ is always empty. Notice that if Γ, Γ' are templates with type $\sigma \to \sigma$ and disjoint support, then $\Gamma \odot \Gamma' = \Gamma' \odot \Gamma$.

Definition 3.31 (Atomic box, atomic template)**.**
A box $\alpha = \langle \overrightarrow{\mathfrak{p}}, G, \overleftarrow{\mathfrak{p}} \rangle \in \mathcal{B}(S,S)$ is *atomic* if its graph has a single non-trivial connected component C, and if for every vertex v outside C, there is a unique port $p \in S$ such that $\overrightarrow{\mathfrak{p}}(p) = \overleftarrow{\mathfrak{p}}(p) = v$. An *atomic template* is a finite template exclusively composed of atomic boxes. ∗

Atomic templates with singleton support can be easily iterated. Indeed, the non-trivial connected component of an atomic box with singleton support is TTSP, and may thus be SP-reduced to a graph with only two vertices, joined by a single edge labelled with some expression e. If we then replace e with e^+, and put the resulting box α' in a template $\alpha^\star := \{\mathrm{id}_\sigma, \alpha'\}$, we easily get $(\!|\alpha^\star|\!) = (\!|\alpha|\!)^\star$. Now if $\Gamma = \{\alpha_1, \ldots, \alpha_n\}$ is an atomic template with singleton support, it must be that every α_i has the same singleton support. We may thus reduce their graphs, yielding expressions $e_1, \ldots, e_i$, and use the label $(e_1 + \cdots + e_i)^+$ to do the same construction.

Lemma 3.32. *The non-trivial connected component of an atomic box of type $\sigma \to \sigma$ always contains a vertex c, such that for every port p mapped inside that component, all paths from $\overrightarrow{\mathfrak{p}}(p)$ to a maximal vertex visit c.* ■

Proof. This is a direct consequence of Lemma 3.1 □

This lemma allows us to split an atomic box into the product $\alpha = \alpha^1 \odot \alpha^2$ of two typed boxes such that $\alpha^2 \odot \alpha^1$ has singleton support. Furthermore if $\mathit{support}(\Gamma) \subseteq \mathit{support}(\alpha)$ then $\{\alpha^2\} \odot \Gamma \odot \{\alpha^1\}$ has singleton support.

Another important property is that the supports of atomic boxes of the same type are either disjoint or comparable:

Lemma 3.33. *For every two atomic boxes $\beta, \gamma \in \mathfrak{B}(\sigma, \sigma)$, exactly one of three things can happen:*

- $\mathit{support}(\beta) \subseteq \mathit{support}(\gamma)$*;*
- $\mathit{support}(\gamma) \subseteq \mathit{support}(\beta)$*;*
- $\mathit{support}(\beta) \cap \mathit{support}(\gamma) = \emptyset$*.* ■

Proof. To prove this result one realises that the support of any atomic box $\beta \in \mathfrak{B}(\sigma, \sigma)$ has to be the set of leaves reachable from some vertex in σ. □

Proposition 3.34. *For any atomic template Γ with type $\sigma \to \sigma$, there is a finite template $\Gamma^\star$ with the same type such that $(\!|\Gamma|\!)^\star = (\!|\Gamma^\star|\!)$.* ■

Proof. Let $\Gamma = \{\alpha_1, \dots, \alpha_n\}$ be an atomic template of type $\sigma \to \sigma$, indexed in such a way that $i < j$ entails either $\mathit{support}(\alpha_i) \subseteq \mathit{support}(\alpha_j)$ or $\mathit{support}(\alpha_i) \cap \mathit{support}(\alpha_j) = \emptyset$. We define $\emptyset^\star = 1_\sigma$. Then for every $k \leqslant n$, we split $\{\alpha_1, \dots, \alpha_{k-1}\}$ into Γ_1 and Γ_2, such that $\mathit{support}(\Gamma_1) \subseteq \mathit{support}(\alpha_k)$ and $\mathit{support}(\Gamma_2) \cap \mathit{support}(\alpha_k) = \emptyset$. We obtain:

$$
\begin{aligned}
(\!|\{\alpha_1, \dots, \alpha_{k-1}, \alpha_k\}|\!)^\star &= (\!|(\Gamma_1 + \alpha_k) + \Gamma_2|\!)^\star && \text{(commutativity)} \\
&= (\!|\Gamma_1|\!) \cup (\!|\alpha_k|\!))^\star \odot (\!|\Gamma_2|\!)^\star && (\mathit{support}(\Gamma_2) \cap \mathit{support}(\Gamma_1 \cup \alpha_k) = \emptyset) \\
&= (\!|\Gamma_1^\star|\!)(\!|\alpha_k \cdot \Gamma_1^\star|\!)^\star \odot (\!|\Gamma_2^\star|\!) && \text{(regular laws)} \\
&= (\!|\Gamma_1^\star|\!)\left((\!|1_\sigma|\!) \cup (\!|\alpha_k \Gamma_1^\star|\!)(\!|\alpha_k \Gamma_1^\star|\!)^\star\right) \odot (\!|\Gamma_2^\star|\!) && \text{(regular laws)} \\
&= (\!|\Gamma_1^\star|\!)\left((\!|1_\sigma|\!) \cup (\!|\alpha_k^1 \alpha_k^2 \Gamma_1^\star|\!)(\!|\alpha_k^1 \alpha_k^2 \Gamma_1^\star|\!)^\star\right) \odot (\!|\Gamma_2^\star|\!) && (\alpha_k = \alpha_k^1 \alpha_k^2) \\
&= (\!|\Gamma_1^\star|\!)\left((\!|1_\sigma|\!) \cup (\!|\alpha_k^1|\!)(\!|\alpha_k^2 \Gamma_1^\star \alpha_k^1|\!)^\star (\!|\alpha_k^2 \Gamma_1^\star|\!)\right) \odot (\!|\Gamma_2^\star|\!) && \text{(regular laws)}
\end{aligned}
$$

As we noticed earlier, because we have $\mathit{support}(\Gamma_1) \subseteq \mathit{support}(\alpha_k)$ the template $\alpha_k^2 \Gamma_1^\star \alpha_k^1$ has a singleton support. This means we can compute its star, thus reduce the last expression into a single finite template. □

3.4 Main theorem

We will now establish the full Kleene Theorem of Petri automata and graph expressions, by extracting expressions from Petri automata.

Theorem 3.35. *The class of the regular sets of graphs coincide with the class of recognisable sets of graphs.* ■

Let us fix an automaton $\mathscr{A} = \langle P, \mathscr{T}, \iota \rangle$. We assume that ι is never in the output of a transition (if it is not the case, it is easy to modify $\mathscr{A}$ to enforce this). We also add a new place $f \notin P$, which is not connected to any transition, but will be useful in the following. For the sake of clarity, we now take the set of places P to be the previous P with the addition of f. We start by building a finite state automaton whose states are types, and transitions are typed boxes. Then, using a procedure similar to the proof of the classical Kleene's Theorem, we reduce it into a single box template from which we can extract a regular graph expression.

3.4.1 A regular language of runs.

We can associate a box to every transition of a proper run, as illustrated in Example 3.38.

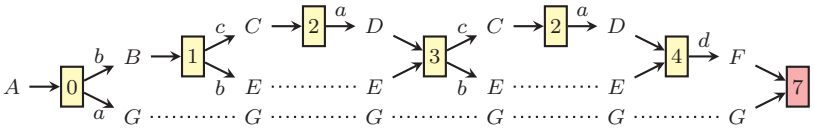
Definition 3.36 (Box of a transition)**.**
Suppose $t \in \mathscr{T}$ is a non-final transition in $\mathscr{A}$, and $S, S' \subseteq P$ are two states such that $\mathscr{A} \vdash t : S \to S'$. The *box of t from S* (written $box\,(t, S)$) is built as follows. Its set of input ports (respectively output ports) is S (resp. S'). The vertices of its graph are the places in S', together with an additional node $*$. Places in ${}^\triangleright t$ are mapped by $\overrightarrow{\mathfrak{p}}$ to $*$, and the others are mapped to themselves. $\overleftarrow{\mathfrak{p}}$ sends every place in S' (seen as a port) to itself (seen as a vertex). Finally, we put edges $\langle *, a, q\rangle$ whenever $\langle a, q\rangle \in \overleftarrow{\mathfrak{p}}$. For final transitions $t = \langle {}^\triangleright t, \emptyset\rangle$ we adapt the construction from state ${}^\triangleright t$ to reach the state $\{f\}$, by defining $box\,(t, {}^\triangleright t)$ to be $\langle [\_ \mapsto *], \langle \{*\}, \emptyset\rangle, [f \mapsto *]\rangle$. $*$

We extend this construction to runs in a straightforward way: if $S_0, \ldots, S_n$ are states and $R = \langle S_0, t_1; \ldots; t_n, S_n\rangle$ is a proper run in $\mathscr{A}$ we define $box\,(R) := box\,(t_1, S_0) \odot box\,(t_2, S_1) \odot \cdots \odot box\,(t_n, S_{n-1})$. With this definition, accepting runs will yield boxes in $\mathcal{B}(\{\iota\}, \{f\})$. This actually amounts to computing the trace of R:

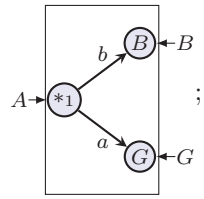
Lemma 3.37. *If $box\,(R) = \langle \overrightarrow{\mathfrak{p}}, G, \overleftarrow{\mathfrak{p}}\rangle$, then G is isomorphic to $\mathscr{G}\,(R)$.* ■

Proof. The vertex k in $\mathscr{G}\,(R)$ (produced by t_k) is equivalent to the vertex $*$ coming from the same transition. □

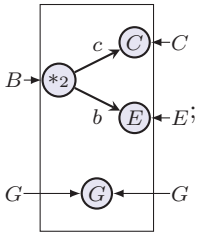
Example 3.38.
Recall the accepting run from Example 3.6:



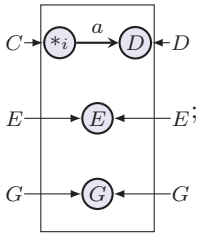
The transitions of this run yield the following boxes:

$box\,(0, \{A\}) =$ 

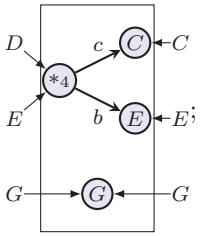
;

$box\,(1, \{B, G\}) =$ 

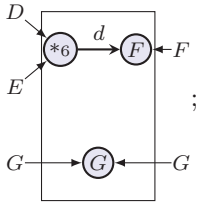
;

$box\,(2, \{C, E, G\})^{\,1} =$ 

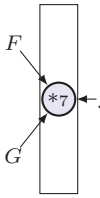
;

$box\,(3, \{D, E, G\}) =$ 

;

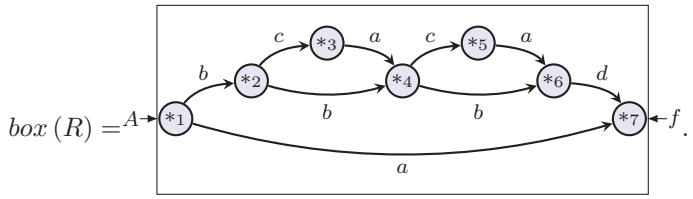
$box\,(4, \{D, E, G\}) =$ 

;

$box\,(7, \{F, G\}) =$ 

.

We may then compute the translation of the run:



The graph of this box is exactly the trace of this run (see Figure 3.6). Notice also that the only vertices in the composite box are the $*_i$ vertices, one for each transition. •

This equivalence with traces yields another property: because of Constraint 2, we know that every trace in $\mathscr{A}$ is TTSP. That means that all the boxes β_i can be typed, starting with the type $\tau_\iota = \circ\!\!\longrightarrow\!\!\circ\!\!-\iota$ and ending with type $\tau_f = \circ\!\!\longrightarrow\!\!\circ\!\!-f$.

Lemma 3.39. *Let $R = \langle \{\iota\}, t_1; \ldots; t_n, \emptyset \rangle$ be an accepting run, with intermediary states $\{\iota\} = S_0, S_1, \ldots, S_{n-1}, S_n = \emptyset$. There exists a sequence $\tau_0, \ldots, \tau_n$ of types over P such that $\tau_0 = \tau_\iota$, $\tau_n = \tau_f$ and $\forall 1 \leqslant i \leqslant n$, box $(t_i, S_{i-1}) \in \mathfrak{B}(\tau_{i-1}, \tau_i)$* ■

Proof. Simply put, if $\mathscr{G}(R)$ is TTSP, then for every k it must be the case that $\mathscr{G}(\langle \{\iota\}, t_1; \ldots; t_k, S_k \rangle)$ SP-reduces to a tree τ_k. Using Lemma 3.37 we can use these trees to types the boxes of every transition in R.

The fact that $\tau_n = \tau_f = \circ\!\!\longrightarrow\!\!\circ\!\!-f$ is quite straightforward, as this is the only type over $\{f\}$. □

Consider the finite state automaton *Aut* with states $\mathbb{T}_P$ (the set of types over P), initial state τ_ι, a single final state τ_f and two kinds of transitions. For every non-final transition t, states S, S' and types τ, σ such that $\mathscr{A} \vdash t : S \to S'$ and box $(t, S) \in \mathfrak{B}(\tau, \sigma)$, there is a transition $\langle \tau, box(t, S), \sigma \rangle$ in *Aut*. There are also transitions $\langle \tau, box(t, {}^\triangleright t), \tau_f \rangle$ for every final transition $\langle {}^\triangleright t, \emptyset \rangle$ and every type τ over ${}^\triangleright t$.

This automaton captures exactly the accepting runs of $\mathscr{A}$. Indeed, Lemma 3.39 assures us that for every accepting run in $\mathscr{A}$ we can find a corresponding accepting run in *Aut*. On the other hand every accepting run in *Aut* stems by construction from an accepting run in $\mathscr{A}$.

This means we could extract a regular expression $e \in \mathrm{Reg}\langle \mathfrak{B} \rangle$ from this automaton using the standard Kleene Theorem. We could then get back the language of $\mathscr{A}$ by extracting the graphs of boxes $\beta_1 \odot \cdots \odot \beta_n$, whenever $\beta_1 \ldots \beta_n$ is a word in $[\![e]\!]$. However, this is not yet what we are looking for: we want a *graph expression over the alphabet* Σ. To get this, we will replay the classic proof of Kleene's theorem, with some modifications to suit our needs.

Remark. The automaton above should be built from the state τ_ι, exploring all initial runs. This construction will fail if either Constraint 1 or Constraint 2 are not satisfied. This means that the two constraints we imposed are decidable.

3.4.2 Computing the expression.

A classical way of proving Kleene's theorem is to move from automata to "generalised automata", that is automata with expressions labelling transitions. We do a similar step here, by replacing boxes with box templates as transition labels. The transformation is straightforward: for every pair of states σ, τ, we put in the generalised automaton a transition labelled with $\{\beta \mid \langle \sigma, \beta, \tau \rangle \in Aut\}$. This ensures that there is exactly one transition

[1] Two instances of this box have to be used, one with $i = 3$ and one with $i = 5$.

between any pair of states. Notice that in the resulting generalised automaton, if the transition from σ to τ is labelled with Γ, all boxes in Γ have type $\sigma \to \tau$, therefore Γ itself has this type.

Then, we proceed to remove non-initial non-final states, in an arbitrary order. Notice that the automaton *Aut* we are considering has a single initial state and a single final state, respectively τ_ι and τ_f. These states are distinct, there is no outgoing transition from τ_f, nor is there an incoming transition in τ_ι (remember that ι does not appear in the output of transitions). At the end of this procedure, the only remaining states will thus be τ_ι and τ_f. There will be exactly one transition from τ_ι to τ_f of the shape $\langle \tau_\iota, \{\beta_1, \cdots, \beta_n\}, \tau_f \rangle$. As any β_i has type $\tau_\iota \to \tau_f$, its graph G_i must be TTSP, thus we get that $e = \mathscr{W}(G_1) + \cdots + \mathscr{W}(G_n)$ is a graph expression and $(\!|\{\beta_1, \cdots, \beta_n\}|\!) \equiv \mathscr{G}(e)$.

We will maintain an invariant throughout the construction: the boxes generated by templates labelling any transition of the automaton should stem from runs in $\mathscr{A}$. More precisely, we require every template labelling a transition to be $\mathscr{A}$-valid:

Definition 3.40 ($\mathscr{A}$-validity)**.**
A template $\gamma \in \mathfrak{BT}(\tau, \sigma)$ is *$\mathscr{A}$-valid* if for every $\beta \in (\!|\gamma|\!)$ there exists a run R in $\mathscr{A}$ such that $box(R) \equiv \beta$. ∗

Now we only need to show how to remove one state, while preserving typing and the language of the automaton. The idea when removing a state τ is to add transitions to replace every run going through τ. For every pair of states σ, χ with transitions labelled with β, δ, γ and Γ' as below, we will define a template Γ. We will then replace Γ' with $\Gamma + \Gamma'$ on the transition going from σ to χ.

We would like Γ to be $\beta \cdot \gamma^\star \cdot \delta$. However, remember that we can only compute the star of atomic templates. But in this case, we can approximate γ with a good-enough atomic template:

Lemma 3.41. *For any $\mathscr{A}$-valid template $\gamma \in \mathfrak{BT}(\tau, \tau)$ there exists an $\mathscr{A}$-valid atomic template $At(\gamma) \in \mathfrak{BT}(\tau, \tau)$ such that $(\!|\gamma|\!) \subseteq (\!|At(\gamma)^\star|\!)$.* ■

Proof. From any connected component of the graph of any box in γ stems an $\mathscr{A}$-valid atomic box. Every box in γ is equal to the product (in any order) of the boxes corresponding to its connected components. We then take $At(\gamma)$ to be the set of all these atomic boxes. □

We then define Γ to be $\beta \cdot At(\gamma)^\star \cdot \delta$. Hence we get $(\!|\beta|\!) \odot (\!|\gamma|\!)^\star \odot (\!|\delta|\!) \subseteq (\!|\Gamma|\!)$. The lemma also ensures that for every graph produced by a run in the automaton where τ is removed, there is a run in $\mathscr{A}$ yielding the same graph. Both theses properties allow us to conclude that this step is valid, preserving both the invariant and the language of the automaton.

3.5 Relationship with Branching Automata

Between 1998 and 2001, Lodaya and Weil introduced another kind of Petri net-based automata called "branching automata" [42, 43, 44]. They obtained a Kleene Theorem for this model, using expressions from $\mathrm{GReg}\langle\Sigma\rangle$ as well. In this section we recall the definition of branching automata and describe precisely the relationship between their result and our own.

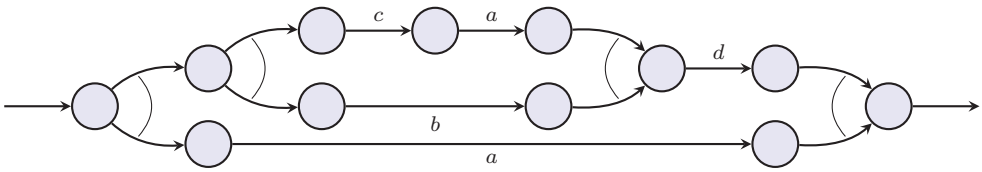


Figure 3.12: Example of branching automaton

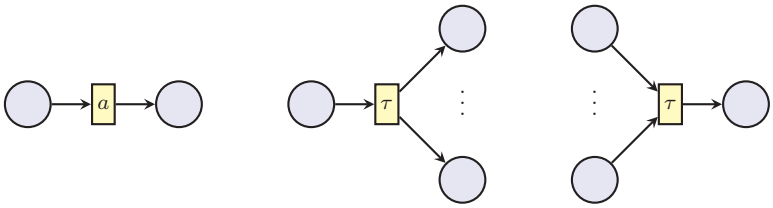


Figure 3.13: Prescribed transitions of branching automata

3.5.1 Definitions and Kleene Theorem

Definition 3.42 (Branching automaton)**.**
A *branching automaton* over the alphabet Σ is a tuple $\langle Q, T_{seq}, T_{fork}, T_{join}, I, F\rangle$, where Q is a set of states, I and F are subsets of Q, respectively the input and output states, and the transitions are split in three sets:

- $T_{seq} \subseteq Q \times \Sigma \times Q$ is the set of sequential transitions;
- $T_{fork} \subseteq Q \times \mathcal{M}_{ns}(Q)$ is the set of opening transitions;
- $T_{seq} \subseteq \mathcal{M}_{ns}(Q) \times Q$ is the set of closing transitions.

(Here $\mathcal{M}_{ns}(Q)$ represents the set of multisets over Q with cardinality at least 2.) ∗

An example of such an automaton is displayed on Figure 3.12. These automata can be seen as labelled Petri nets of a particular shape: transitions are restricted to the three types described on Figure 3.13.

These automata run on terms over the signature $\langle \Sigma, \cdot, \cap\rangle$, quotiented by associativity of both operations and by commutativity of $\cap$.

Definition 3.43 (Runs and language of a branching automaton)**.**
Let t be a term, there is a *run* on t from state p to state q if:

- $t = a \in \Sigma$ and $\langle p, a, q\rangle \in T_{seq}$;
- $t = t_1 \cap \cdots \cap t_n$ with $n \geqslant 2$, there are two transitions $\langle p, [p_1, \ldots, p_n]\rangle \in T_{fork}$ and $\langle [q_1, \ldots, q_n], q\rangle \in T_{join}$, and for every $1 \leqslant i \leqslant n$ there is a run on t_i from p_i to q_i;
- $t = t_1 \cdot \cdots \cdot t_n$ with $n \geqslant 2$, there are states $p = p_0, p_1, \ldots, p_n = q$, and for every $0 \leqslant i < n$ there is a run on t_i from p_i to p_{i+1}.

The *language* of a branching automaton $\mathscr{B}$, written $\mathscr{L}(\mathscr{B})$ is then defined as the set of terms t such that there exists a pair of states $\langle q_i, q_f\rangle \in I \times F$ such that there is a run on t in $\mathscr{B}$ from q_i to q_f. ∗

We can also associate a set of such terms to any expressions in $\mathrm{GReg}\langle\Sigma\rangle$.

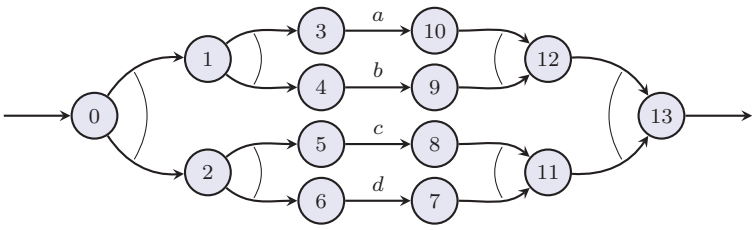


Figure 3.14: Example of branching automaton

Definition 3.44 (Term language of an expression)**.**
The term language of an expression $e \in \mathrm{GReg}\langle\Sigma\rangle$, denoted by $\mathscr{L}(e)$ is defined by induction as follows:

$$\mathscr{L}(a) := \{a\} \qquad \mathscr{L}(0) := \emptyset \qquad \mathscr{L}(e \cup f) := \mathscr{L}(e) \cup \mathscr{L}(f)$$

$$\mathscr{L}(e \cdot f) := \{u \cdot v \mid u \in \mathscr{L}(e), v \in \mathscr{L}(f)\} \qquad \mathscr{L}(e \cap f) := \{u \cap v \mid u \in \mathscr{L}(e), v \in \mathscr{L}(f)\}$$

$$\mathscr{L}\left(e^{+}\right) := \{u_1 \cdot \dots \cdot u_n \mid n > 0, \forall i, u_i \in \mathscr{L}(e)\}.$$

*

To make these two classes of term languages coincide, Lodaya and Weil impose some restrictions on the runs of the automaton, that corresponds to a safety constraint over the underlying Petri net, much like Constraint 1. Here we only consider branching automata implicitly satisfying those constraints.

Theorem 3.45 (Kleene Theorem for branching automata)**.** *For every set of terms $\mathcal{T}$ the following are equivalent:*

(i) there is an expression $e \in \mathrm{GReg}\langle\Sigma\rangle$ such that $\mathcal{T} = \mathscr{L}(e)$;

(ii) there is a branching automaton $\mathscr{B}$ such that $\mathcal{T} = \mathscr{L}(\mathscr{B})$. ■

3.5.2 Comparison with Petri automata

At first glance the two Kleene theorems and the fact that both branching automata (BA) and Petri automata (PA) are Petri net-based seem to mean they are completely equivalent. Indeed the same set of regular-like expressions may be used to describe their semantics. However they still exhibit some deep differences.

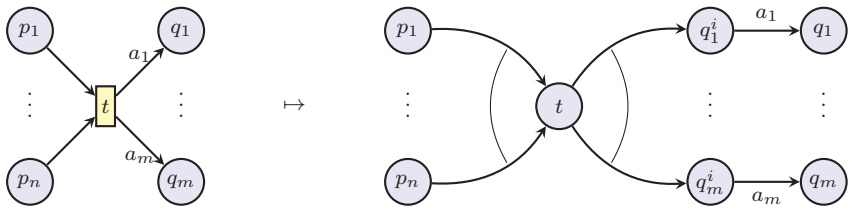
The first difference comes from the runs in the two models. In some sense, the runs in BA require a "global" view of the term being read. Consider the BA in Figure 3.14, and the term $t = b \cap c \cap a \cap d$. t is accepted by this automaton, but in order to reach that conclusion, one must: 1) refactor t as $(a \cap b) \cap (c \cap d)$; 2) "match" the opening transition $\langle 0, [1, 2]\rangle$ with the closing transition $\langle [11, 12], 13\rangle$; 3) read the subterms $(a \cap b)$ and $(c \cap d)$ respectively from state 1 to state 12 and from 2 to 11. This means that the run is built as a nesting of runs (rather than a sequential process), and that it needs to manipulate the term as a whole (rather than using a partial, local view of it).

By contrast, to fire a transition in a PA, one simply needs to see one vertex of the graph and the edges coming out of it. Furthermore, the matching of transitions is somewhat automatic in our model, and knowledge of it is not needed to compute a run. For these reasons, our notion of language of a run could be defined using standard Petri net notions (namely pomset-traces, see Section 4.7 for more details), whereas runs in a branching automaton rely crucially on the use of terms.

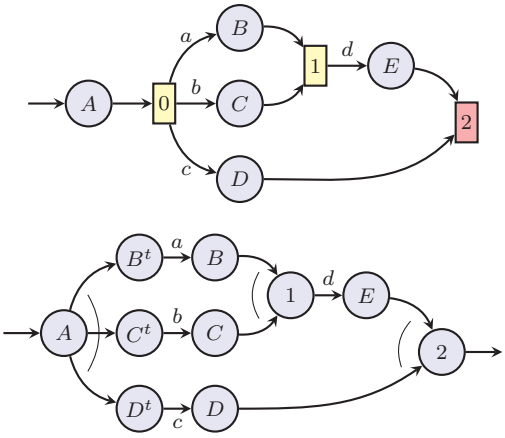
Another difference stems from the way the automata are labelled. BA do not label their opening and closing transitions, making these "silent transitions". This complicates greatly the task of comparing automata using simulation-based methods. In our case though, every transition is labelled, allowing use to define an algorithm to compare automata (see Chapter 4).

These differences make the task of converting from one model to the other rather subtle. To translate a BA into a PA, one would need some kind of epsilon elimination procedure. We believe such a procedure could be devised using boxes to keep track of the order in which opening and closing transitions are combined. However we do not have a precise formulation of this algorithm yet.

The other direction is slightly easier. First, one modifies the transitions as sketched below:



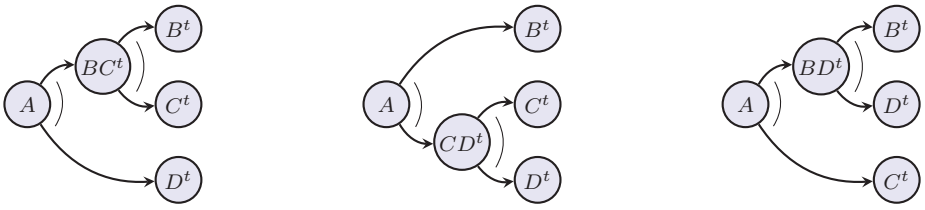
This yields an automaton whose pomset-trace language (when seen as a Petri net) is the language of the original PA. However, it's branching automaton language is not the same, as illustrated by the following example:



The pomset-trace language of this branching automaton corresponds to the set $\{((a \cap b) \cdot d) \cap c\}$, but its language is empty, as no factorisation of that term has three parallel subterms. To solve this problem, one need to saturate the automaton by splitting opening and closing transitions to allow for every factorisation. Formally, it means that for an opening transition $\langle p, [q_1, \ldots, q_n] \rangle$, for every tree[2] with n leaves labelled with $[q_1, \ldots, q_n]$ there should be a sequence of opening transitions of that shape.

In the case of the above example, one would add three states BC^t, BD^t and CD^t, and six transitions:

[2]We consider here trees where no vertex has out-degree one.



This procedure will allow for the pomset-trace and the BA languages to coincide, thus producing a BA with the same language as the original PA.

This translation gives rise to a different proof of Theorem 3.35: first translate the PA into an equivalent BA, and use Theorem 3.45 to obtain an expression describing its language. Conversely, one could obtain Theorem 3.45 through the translation from BA to PA and Theorem 3.35.

We found out about this work by Lodaya and Weil after having proved our theorem. Nevertheless we feel that the tools and techniques we developed to obtain this result bring an interesting new point of view on the matter, and could prove useful in the future.

3.6 Conclusion

In this chapter we provided two ways of defining sets of series-parallel graphs: either as the language $\mathscr{G}(e)$ of some expression e, or as the set $\mathscr{G}(\mathscr{A})$ of traces of a Petri automaton $\mathscr{A}$. These definitions are very different: $\mathscr{G}(e)$ is defined inductively in a compositional manner, which makes is suited to equational reasoning, whereas $\mathscr{G}(\mathscr{A})$ has a more operational flavour, making it better suited for complexity analysis and algorithmic manipulation. With Theorem 3.35 however, we showed that we can define the same sets of graphs using both methods.

A lot of research has been done in this area, yielding results similar to our own, but with different perspectives and very different methods. For instance Courcelle et al. [23] developed a complete algebraic language to describe graphs. However it seems that they use quite a different approach to describe sets of graph: in our case sets of graphs are inductively generated from expressions, whereas Courcelle tends to use the set of solutions of some equation.

The results by Bossut, Dauchet and Warin [10] could also be relevant here, as they provide a Kleene theorem for a class of graphs very similar to our own. The main difficulty in relating this work to our own stems from the very different style of automata and expressions used.

Fourth chapter

Petri automata for Kleene Allegories

"I dislike Allegory – the conscious and intentional allegory – yet any attempt to explain the purport of myth or fairytale must use allegorical language."

— J. R. R. Tolkien.

4.1 Introduction

We consider in this chapter binary relations and the operations of union ($\cup$), intersection ($\cap$), composition ($\cdot$), converse ($\_^\smile$), transitive closure ($\_^+$), reflexive-transitive closure ($\_^\star$), and the constants identity (1), empty relation (0) and universal relation ($\top$). This model gives rise to an (in)equational theory: a pair of terms e, f made from those operations and some variables $a, b, \ldots$ is a *valid equation*, denoted $\mathcal{R}el \models e = f$, if the corresponding equality holds universally. Similarly, an inequation $\mathcal{R}el \models e \leq f$ is valid when the corresponding containment holds universally. Here are valid equations and inequations: they hold whatever the relations we assign to variables a, b, and c.

$$\mathcal{R}el \models (a \cup b)^\star \cdot b \cdot (a \cup b)^\star = (a^\star \cdot b \cdot a^\star)^+ \tag{4.1}$$

$$\mathcal{R}el \models a^\star \leq 1 \cup a \cdot a^\smile \cdot a^+ \tag{4.2}$$

$$\mathcal{R}el \models a \cdot b \cap c \leq a \cdot (b \cap a^\smile \cdot c) \tag{4.3}$$

$$\mathcal{R}el \models a^+ \cap 1 \leq (a \cdot a)^+ \tag{4.4}$$

Various fragments of this theory have been studied in the literature:

- *Kleene algebra* [22], where one removes intersection, converse, and $\top$, so that terms are plain regular expressions. The equational theory is decidable [35], and actually PSpace-complete [46]. The equational theory is not finitely based [55], but finite quasi-equational axiomatisations exist [41, 36]. Equation (4.1) lies in this fragment, and one can notice that the two expressions recognise the same language.
- *Kleene algebra with converse*, where one only removes intersection and $\top$, is also a decidable fragment [6]. It remains PSpace [15]. Inequation (4.2) belongs to this fragment; it can be axiomatised relatively to Kleene algebra [26].
- (representable, distributive) *allegories* [27], sometimes called positive relation algebras, where transitive and reflexive-transitive closures are not allowed. They are decidable [27, page 208]; and not finitely based. Inequation (4.3) is known as the *modularity law* in this setting.

To the best of our knowledge, the decidability of the whole theory, *bounded Kleene allegories*, is open. Here we obtain several important steps towards the resolution of this problem:

1. we give a characterisation of the full (in)equational theory in terms of graph languages, and we relate this to the construction from the previous chapter;
2. we design a way to recognise such graph languages with Petri automata;
3. we show how to associate such a graph automaton to any term of Kleene allegories;
4. using these graph automata, we give a decision procedure for the fragment where the maximal relation, converse and identity are forbidden.

The latter fragment was studied recently by Andréka et al. [3]; its decidability was open as far as we know. The restriction to this fragment allows us to exploit simplifying assumptions about the produced automata, and to obtain a coinductive algorithm for language inclusion (Section 4.5). We actually show that language inclusion for these automata is ExpSpace-complete (Section 4.6).

The next problem, which remains open, consists in obtaining the decidability of language inclusion in the full automata model: together with the presented results, this would entail decidability of Kleene allegories. We outline some of the difficulties arising with converse or unit in the presence of intersection in Section 4.5.4.

4.2 Graphs and expressions

We consider the following set of expressions:

Definition 4.1 (Allegoric regular expressions)**.**
Allegoric regular expressions over Σ, or simply expressions in the remaining of this chapter, are terms $e, f \dots$ built on the signature $\langle 0, 1, \top, \cup, \cdot, \cap, \_^\smile, \_^\star \rangle$. We denote by $\mathrm{AReg}\langle\Sigma\rangle$ the set of expressions over the finite alphabet Σ. ∗

Ground terms are the expressions $u, v, w \dots$ built with the sub-signature $\langle \cap, \cdot, \_^\smile, 1, \top \rangle$. The set of ground terms over Σ is written $\mathbb{W}_\Sigma$. If $\sigma : \Sigma \to \mathcal{R}el\langle S \rangle$ is an interpretation of the alphabet Σ into some algebra of relations, we write $\widehat{\sigma}$ for the unique homomorphism extending σ into a function from $\mathrm{AReg}\langle\Sigma\rangle$ to $\mathcal{R}el\langle S \rangle$. An inequation between two expressions e and f is *valid*, written $\mathcal{R}el \models e \leq f$, if for any relational interpretation σ we have $\widehat{\sigma}(e) \subseteq \widehat{\sigma}(f)$.

We let G range over 2-pointed labelled directed graphs[1], which we simply call *graphs* in the sequel. Those are tuples $\langle V, E, \iota, o \rangle$ with V a finite set of vertices, $E \subseteq V \times \Sigma \times V$ a set of edges labelled with Σ, and $\iota, o \in V$ the two distinguished vertices, respectively called *input* and *output*.

Definition 4.2 (Graph of a ground term: $\mathcal{G}(w)$)**.**
To each ground term w, we associate a graph $\mathcal{G}(w)$, by induction on w. The graph of $a \in \Sigma$ has one edge labelled by a linking its input to its output. The graph for 1 has only one vertex, both input and output. The graph of $\top$ has an input distinct from its output, without any edge[2]. The composition of two graphs with disjoint sets of vertices can be performed by identifying the output of the first graph and the input of the second one. The intersection on graphs consists in merging their inputs and merging their outputs. The converse consists simply in exchanging the input and the output of a graph. ∗

See Figure 4.1 for a graphical description of this construction. Those graphs were introduced independently by Freyd and Scedrov [27, page 208], and Andréka and Bredikhin [2]. Notice that the constructions for variables, products and intersections are the same as in the previous chapter. We investigate this relationship in Section 4.3.

Definition 4.3 (Graph homomorphism, preorders $\blacktriangleleft$ and $\lhd$)**.**
A *graph homomorphism* from $\langle V_1, E_1, \iota_1, o_1 \rangle$ to $\langle V_2, E_2, \iota_2, o_2 \rangle$ is a map $\varphi : V_1 \to V_2$ such that $\varphi(\iota_1) = \iota_2$, $\varphi(o_1) = o_2$, and $\langle p, x, q \rangle \in E_1$ entails $\langle \varphi(p), x, \varphi(q) \rangle \in E_2$. We denote by $\blacktriangleleft$ the relation on graphs defined by $G \blacktriangleleft G'$ if there exists a graph homomorphism from G' to G. This relation gives rise to a preorder on ground terms, written $\lhd$ and defined by $u \lhd v$ if $\mathcal{G}(u) \blacktriangleleft \mathcal{G}(v)$. ∗

[1] Notice that in contrast with the previous chapter, we do not require graphs to be acyclic here.

[2] The author would like to thank Florent Bréhard for the idea of the graph of $\top$. It's so simple once you have the good definition!

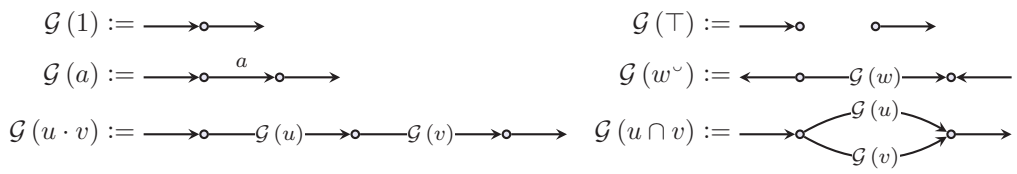


Figure 4.1: Inductive construction of the graph of a ground term.

Given a set S of graphs, we write ${}^{\blacktriangleleft}S$ for its downward closure: ${}^{\blacktriangleleft}S := \{G \mid G \blacktriangleleft G', G' \in S\}$. Similarly, we write ${}^{\triangleleft}S$ for the downward closure of a set of ground terms w.r.t. $\triangleleft$.

A significant number of the results presented in [2, 3] rely on [2, Lemma 3]. We restate it here with $\top$ added to the signature of ground terms.

Lemma 4.4. *Let S be a base set, $i, j \in S$, $v \in \mathbb{W}_\Sigma$, $\mathcal{G}(v) = \langle V_v, E_v, \iota_v, o_v\rangle$ and σ a map from Σ to $\mathcal{R}el\,\langle S\rangle$. Then $\langle i, j\rangle \in \widehat{\sigma}(v)$ if and only if there exists a function $\varphi : V_v \to S$ such that $\varphi(\iota_v) = i$; $\varphi(o_v) = j$ and if $\langle p, a, q\rangle \in E_v$ then $\langle \varphi(p), \varphi(q)\rangle \in \sigma(a)$.* ■

Proof. Because this lemma was stated without $\top$, we formally need to reprove it in our context. Fortunately, the proof in [2] relies on an induction, hence we may reuse their proof, only looking at the case $v = \top$, which holds trivially. □

The above preorder on ground terms precisely characterises inclusion under arbitrary relational interpretations:

Theorem 4.5. *For all ground terms u, v, we have $\mathcal{R}el \models u \leq v \Leftrightarrow u \lhd v$.* ■

Proof. This lemma already exists as [2, Theorem 1], or [27, page 208], but without the $\top$ operation. As the proof of [2, Theorem 1] mainly relies on [2, Lemma 3], we may replay it word for word, deferring to Lemma 4.4 instead. □

To extend this result to bounded Kleene allegories, we introduce the following generalisation of the language of a regular expression. Sets of words become sets of ground terms, and sets of graphs.

Definition 4.6 (Terms of an expression)**.**
The *set of terms* of an expression $e \in \mathrm{AReg}\,\langle\Sigma\rangle$, written $[\![e]\!]$, is the set of ground terms defined inductively as follows:

$$\begin{array}{rrr} [\![1]\!] := \{1\} & [\![\top]\!] := \{\top\} & [\![0]\!] := \emptyset \\ [\![a]\!] := \{a\} & [\![e \cdot f]\!] := \{w \cdot w' \mid w \in [\![e]\!] \text{ and } w' \in [\![f]\!]\} & \\ [\![e \cup f]\!] := [\![e]\!] \cup [\![f]\!] & [\![e \cap f]\!] := \{w \cap w' \mid w \in [\![e]\!] \text{ and } w' \in [\![f]\!]\} & \\ [\![e^\smile]\!] := \{w^\smile \mid w \in [\![e]\!]\} & [\![e^\star]\!] := \bigcup_{n\in\mathbb{N}} \{w_1 \cdot \dots \cdot w_n \mid \forall i, w_i \in [\![e]\!]\} \ . & * \end{array}$$

Notice that if $e \in \mathrm{AReg}\,\langle\Sigma\rangle$ is a regular expression (meaning it never uses the extra operators $\top$,$^\smile$ and $\cap$), $[\![e]\!]$ is the rational language denoted by e.

Definition 4.7 (Graphs of an expression)**.**
The *set of graphs* of an expression $e \in \mathrm{AReg}\,\langle\Sigma\rangle$, written $\mathfrak{G}(e)$, is the set of graphs defined inductively as follows:

$$\begin{array}{rrr} \mathfrak{G}(1) := \{\mathcal{G}(1)\} & \mathfrak{G}(\top) := \{\mathcal{G}(\top)\} & \mathfrak{G}(0) := \emptyset \\ \mathfrak{G}(a) := \{\mathcal{G}(a)\} & \mathfrak{G}(e \cdot f) := \{G \cdot G' \mid G \in \mathfrak{G}(e) \text{ and } G' \in \mathfrak{G}(f)\} & \\ \mathfrak{G}(e \cup f) := \mathfrak{G}(e) \cup \mathfrak{G}(f) & \mathfrak{G}(e \cap f) := \{G \cap G' \mid G \in \mathfrak{G}(e) \text{ and } G' \in \mathfrak{G}(f)\} & \\ \mathfrak{G}(e^\smile) := \{G^\smile \mid G \in \mathfrak{G}(e)\} & \mathfrak{G}(e^\star) := \bigcup_{n\in\mathbb{N}} \{G_1 \cdot \dots \cdot G_n \mid \forall i, G_i \in \mathfrak{G}(e)\} \ . & * \end{array}$$

It is a simple matter to check that $\mathfrak{G}(e) = \{\mathcal{G}(w) \mid w \in [\![e]\!]\}$.

To obtain the characterisation announced in the introduction, we need a slight refinement of a lemma established by Andréka, Mikulás, and Németi [3]:

Lemma 4.8. *For any expression $e \in AReg\langle\Sigma\rangle$, any set S and any relational interpretation $\sigma : \Sigma \to \mathcal{R}el\langle S\rangle$, we have*

$$\widehat{\sigma}(e) = \bigcup_{u \in [\![e]\!]} \widehat{\sigma}(u) = \bigcup_{w \in {}^{\lhd}[\![e]\!]} \widehat{\sigma}(w). \qquad \blacksquare$$

Proof. The first equality is [3, Lemma 2.1] (extended with $\top$); for the second one, we use the fact that $\widehat{\sigma}(w) \subseteq \widehat{\sigma}(u)$ whenever $w \lhd u$, thanks to Theorem 4.5. □

Theorem 4.9. *The following properties are equivalent, for all expressions $e, f \in AReg\langle\Sigma\rangle$:*

(i) $\mathcal{R}el \models e \leq f$, *(ii)* $[\![e]\!] \subseteq {}^{\lhd}[\![f]\!]$, *(iii)* $\mathfrak{G}(e) \subseteq {}^{\blacktriangleleft}\mathfrak{G}(f)$. $\blacksquare$

Proof. The implication (ii) ⇒ (i) follows easily from Lemma 4.8, and (iii) ⇒ (ii) is a matter of unfolding definitions. For (i) ⇒ (iii), we mainly use Lemma 4.4:

Let $e, f \in \mathrm{AReg}\langle\Sigma\rangle$ two expressions such that $\mathcal{R}el \models e \leq f$, and $u \in [\![e]\!]$ such that $\mathcal{G}(u) = \langle V_u, E_u, \iota_u, o_u\rangle$; we can build an interpretation $\sigma : \Sigma \to \mathcal{R}el\langle V_u\rangle$ by specifying:

$$\sigma(a) := \{\langle p, q\rangle \mid \langle p, a, q\rangle \in E_u\}.$$

It is quite simple to check that $\widehat{\sigma}(u) = \{\langle \iota_u, o_u\rangle\}$. By Lemma 4.8 and $\mathcal{R}el \models e \leq f$, we know that

$$\widehat{\sigma}(u) \subseteq \widehat{\sigma}(f) = \bigcup_{v \in [\![f]\!]} \widehat{\sigma}(v).$$

Thus there is some $v \in [\![f]\!]$ such that $\langle \iota_u, o_u\rangle \in \widehat{\sigma}(v)$. By Lemma 4.4 we get that there is a map $\varphi : V_v \to V_u$ such that $\varphi(\iota_v) = \iota_u$; $\varphi(o_v) = o_u$ and

$$\langle p, a, q\rangle \in E_v \Rightarrow \langle \varphi(p), \varphi(q)\rangle \in \sigma(a).$$

Using the definition of σ, we rewrite this last condition as

$$\langle p, a, q\rangle \in E_v \Rightarrow \langle \varphi(p), a, \varphi(q)\rangle \in E_u.$$

Thus φ is a graph homomorphism from $\mathcal{G}(v)$ to $\mathcal{G}(u)$, proving that $\mathcal{G}(u) \blacktriangleleft \mathcal{G}(v)$, hence $\mathcal{G}(u) \in {}^{\blacktriangleleft}\mathfrak{G}(f)$. □

A simple corollary of this theorem is that in order to decide the validity of the law $e \leq f$, we "only" have to test whether ${}^{\blacktriangleleft}\mathfrak{G}(e) \subseteq {}^{\blacktriangleleft}\mathfrak{G}(f)$. Similarly, if we are interested in $e = f$, we can test if ${}^{\blacktriangleleft}\mathfrak{G}(e) = {}^{\blacktriangleleft}\mathfrak{G}(f)$ holds. In Section 4.4 we will show how Petri automata can be used to describe the languages ${}^{\blacktriangleleft}\mathfrak{G}(e)$.

A connection with Kleene algebra with converse.

It is interesting to notice that the order $\lhd$ we define here is strongly related with the reduction relation we defined in Chapter 2, Definition 2.6.

Remember that we associated with every word u over the alphabet $\mathbf{X} = X \cup \{x' \mid x \in X\}$ a function $\widehat{\varphi_u} : \mathbf{X}^\star \to \mathcal{R}el\langle\{0, \ldots, |u|\}\rangle$ satisfying the equation:

$$\langle 0, n\rangle \in \widehat{\varphi_u}(v) \Leftrightarrow v \rhd^\star u \ . \tag{2.13}$$

Notice that the graphs of words over $\mathbf{X}$ are isomorphic to the words themselves. Using (2.13) and Lemma 4.4, we obtain that for all words $u, v \in \mathbf{X}^\star$,

$$v \rhd^\star u \ \Leftrightarrow \ u \lhd v \ . \tag{4.5}$$

This is consistent with [6]: homomorphisms between such linear graphs are precisely what Bloom et al. define as the set of *admissible functions* γ from the prefixes of the word v to the prefixes of u such that $\gamma(v) = u$ and they show that $v \rhd^* u$ if and only if there exists such a map [6, Proposition 5.13]. This can be extended to regular expressions with converse, and we get that the languages ${}^{\lhd}[\![\mathbf{e}]\!]$ and ${}^{\lhd}[\![e]\!]$ are isomorphic.

4.3 From allegoric expressions to graph expressions

We may relate these graph languages to those we defined in the previous chapter.

Recall (Chapter 2) that one can always see an expression with converse as an expression without converse on a duplicated alphabet. We may also remove $\top$ and 1 from our signature in the same fashion. We do this in two steps.

Definition 4.10 (Converse-normal form)**.**
An expression $e \in \mathrm{AReg}\,\langle\Sigma\rangle$ is in *converse-normal form* if the converse operator is only applied to letters in e. ∗

By using the following rewriting system, we associate with every expression a converse-normal expression denoting the same set of graphs.

$$\begin{array}{lll} (a \cup b)^\smile \to a^\smile \cup b^\smile & 0^\smile \to 0 & (a^\star)^\smile \to (a^\smile)^\star \\ (a \cdot b)^\smile \to b^\smile \cdot a^\smile & 1^\smile \to 1 & a^{\smile\smile} \to a \\ (a \cap b)^\smile \to a^\smile \cap b^\smile & \top^\smile \to \top & \end{array}$$

Definition 4.11 (Translation between allegoric and graph expressions)**.**
Let $e \in \mathrm{AReg}\,\langle\Sigma\rangle$ be an expression in converse-normal form. Consider the following alphabet:

$$\Sigma_\bullet := \Sigma \cup \{a' \mid a \in \Sigma\} \cup \{1, \top\}\,.$$

We define the translated of e to be the expression $\lfloor e \rfloor \in \mathrm{GReg}\,\langle\Sigma_\bullet\rangle$ obtained by replacing $a^\smile$ with a', $e^\star$ with $e^+ \cup 1$, and seeing 1 and $\top$ as letters. ∗

Now we can use the graph construction from the previous chapter with expressions from $\mathrm{GReg}\,\langle\Sigma_\bullet\rangle$, thus yielding two-terminal series parallel graphs labelled with $\Sigma_\bullet$. Hence we simply need to provide a way to transform such graphs into graphs labelled with Σ, with distinguished inputs and outputs. This is the purpose of the function $\lceil G \rceil$:

Definition 4.12 ($\lceil G \rceil$)**.**
Let $G = \langle V, E\rangle$ be a TTSP graph labelled with $\Sigma_\bullet$. Let $\equiv_G$ be the smallest equivalence relation on V containing all pairs $\langle i, j\rangle$ such that $\langle i, 1, j\rangle \in E$, and $[i]_G$ be the equivalence class of i. Then $\lceil G \rceil$ is the graph defined as $\langle V/\equiv_G, E', [\iota]_G, [o]_G\rangle$ where ι is the source of G, o is its sink, and:

$$E' := \{\langle [i]_G, x, [j]_G\rangle \mid x \in \Sigma \text{ and } \exists\, \langle k, l\rangle \in [i]_G \times [j]_G : \langle k, x, l\rangle \in E \text{ or } \langle l, x', k\rangle \in E\}\,. \quad *$$

Lemma 4.13. *Let G_1, G_2 be a pair of TTSP graphs over $\Sigma_\bullet$. We pose $G_k = \langle V_k, E_k\rangle$ ($k \in \{1, 2\}$), its source and sink being denoted respectively by ι_k and o_k. The following hold:*

$$\begin{aligned} \lceil G_1 \rceil \cdot \lceil G_2 \rceil &= \lceil G_1 \cdot G_2 \rceil \\ \lceil G_1 \rceil \cap \lceil G_2 \rceil &= \lceil G_1 \cap G_2 \rceil\,. \end{aligned}$$ ■

$$\begin{array}{ccc} \mathrm{TTSP}\langle\Sigma_\bullet\rangle \times \mathrm{TTSP}\langle\Sigma_\bullet\rangle & \xrightarrow{\lceil\rceil \times \lceil\rceil} & \mathrm{Graph}\langle\Sigma\rangle \times \mathrm{Graph}\langle\Sigma\rangle \\ \downarrow{\scriptstyle \cdot/\cap} & & \downarrow{\scriptstyle \cdot/\cap} \\ \mathrm{TTSP}\langle\Sigma_\bullet\rangle & \xrightarrow[\lceil\rceil]{} & \mathrm{Graph}\langle\Sigma\rangle \end{array}$$

Proof. For the first equation, notice that every path (directed or not) in $G_1 \cdot G_2$ that joins two vertices i, j such that $i \in V_1$ and $j \in V_2$ must visit the vertex o_1/ι_2. This means that we can represent the equivalence classes of $\equiv_{G_1 \cdot G_2}$ as:

$$\{[i]_{G_1} \mid i \in V_1 \setminus [o_1]_{G_1}\} \cup \{[i]_{G_2} \mid i \in V_2 \setminus [\iota_2]_{G_2}\} \cup \{[o_1]_{G_1} \cup [\iota_2]_{G_2}\}.$$

This happens to be exactly the vertices of $\lceil G_1 \rceil \cdot \lceil G_2 \rceil$. We deduce easily from this the equality of the sets of edges and that of the inputs and outputs.

The second equation is established in a similar manner. We notice here that every path in $G_1 \cap G_2$ that goes from V_1 to V_2 either visits ι_1/ι_2 or o_1/o_2. We may thus represent the equivalence classes of $\equiv_{G_1 \cap G_2}$ as:

$$\{[i]_{G_k} \mid k \in \{1,2\},\ i \in V_k \setminus ([\iota_k]_{G_k} \cup [o_k]_{G_k})\} \cup \{[\iota_1]_{G_1} \cup [\iota_2]_{G_2}, [o_1]_{G_1} \cup [o_2]_{G_2}\}.$$

Again, this is the set of vertices of $\lceil G_1 \rceil \cap \lceil G_2 \rceil$, and it is immediate to check the equality from here. □

Proposition 4.14. *If $e \in AReg\langle\Sigma\rangle$ is in converse-normal form, $\mathfrak{G}(e) = \lceil \mathscr{G}(\lfloor e \rfloor) \rceil$.* ■

Proof. We perform a structural induction on e. The base cases, namely $0, 1, \top, a$ and $a^\smile$, are immediate to check by merely unfolding the various definitions. The case of $e \cup f$ doesn't hold any difficulty either:

$$\begin{aligned}\lceil \mathscr{G}(\lfloor e \cup f \rfloor) \rceil &= \lceil \mathscr{G}(\lfloor e \rfloor \cup \lfloor f \rfloor) \rceil = \lceil \mathscr{G}(\lfloor e \rfloor) \cup \mathscr{G}(\lfloor f \rfloor) \rceil = \lceil \mathscr{G}(\lfloor e \rfloor) \rceil \cup \lceil \mathscr{G}(\lfloor f \rfloor) \rceil \\ &= \mathfrak{G}(e) \cup \mathfrak{G}(f) = \mathfrak{G}(e \cup f)\end{aligned}$$

For the case of the sequential and parallel products, as one would expect, Lemma 4.13 will prove useful:

$$\begin{aligned}\mathfrak{G}(e \cdot f) &= \{G \cdot G' \mid \langle G, G' \rangle \in \mathfrak{G}(e) \times \mathfrak{G}(f)\} \\ &= \{\lceil G \rceil \cdot \lceil G' \rceil \mid \langle G, G' \rangle \in \mathscr{G}(\lfloor e \rfloor) \times \mathscr{G}(\lfloor f \rfloor)\} \\ &= \{\lceil G \cdot G' \rceil \mid \langle G, G' \rangle \in \mathscr{G}(\lfloor e \rfloor) \times \mathscr{G}(\lfloor f \rfloor)\} \\ &= \{\lceil G \rceil \mid G \in \mathscr{G}(\lfloor e \cdot f \rfloor)\} = \lceil \mathscr{G}(\lfloor e \cdot f \rfloor) \rceil \\ \mathfrak{G}(e \cap f) &= \{G \cap G' \mid \langle G, G' \rangle \in \mathfrak{G}(e) \times \mathfrak{G}(f)\} \\ &= \{\lceil G \rceil \cap \lceil G' \rceil \mid \langle G, G' \rangle \in \mathscr{G}(\lfloor e \rfloor) \times \mathscr{G}(\lfloor f \rfloor)\} \\ &= \{\lceil G \cap G' \rceil \mid \langle G, G' \rangle \in \mathscr{G}(\lfloor e \rfloor) \times \mathscr{G}(\lfloor f \rfloor)\} \\ &= \{\lceil G \rceil \mid G \in \mathscr{G}(\lfloor e \cap f \rfloor)\} = \lceil \mathscr{G}(\lfloor e \cap f \rfloor) \rceil\end{aligned}$$

The case of $e^\star$ then follows simply from the cases of $\cup$ and $\cdot$. □

4.4 Petri automata

In this section we implicitly consider converse normal expressions without loss of generality.

Proposition 4.14 allows us to associate with every allegorical expression a Petri automaton whose graph language is closely related to the set of graphs of the expression.

Lemma 4.15. *For all $e \in AReg\langle\Sigma\rangle$, $\mathfrak{G}(e) = \lceil \mathscr{G}(\mathscr{A}\lfloor e \rfloor) \rceil$.* ■

Proof. This is immediate using Proposition 4.14 and Lemma 3.16. □

However, remember that the set of graphs we are interested in is not directly $\mathfrak{G}(e)$, but rather ${}^\blacktriangleleft\mathfrak{G}(e)$. We show how to describe directly from a Petri automaton $\mathscr{A}$ the set ${}^\blacktriangleleft\lceil \mathscr{G}(\mathscr{A}) \rceil$. This is done by giving a way to read a graph with an automaton:

Definition 4.16 (Reading, language of a run)**.**
A *reading* of $G = \langle V, E, \iota, o \rangle$ along a run $R = \langle S_0, t_0; \dots; t_n, S_{n+1} \rangle$ (with intermediary states $S_1, \dots$) is a sequence $(\rho_k)_{0 \leqslant k \leqslant n+1}$ such that for all k, ρ_k is a map from S_k to V, $\rho_0(S_0) = \{\iota\}$, and $\forall 0 \leqslant k \leqslant n$, the following holds:

- all tokens in the input of the transition are mapped to the same vertex in the graph:

$$\forall p, q \in {}^{\triangleright}t_k,\ \rho_k(p) = \rho_k(q);$$

- a final transition may only be fired from the output of a graph:

$${}^{\triangleleft}t_k = \emptyset \Rightarrow \forall p \in {}^{\triangleright}t_k, \rho_k(p) = o;$$

- the images of tokens in S_{k+1} that are not in the input of the transition are unchanged:

$$\forall p \in S_{k+1} \setminus {}^{\triangleright}t_k,\ \rho_k(p) = \rho_{k+1}(p);$$

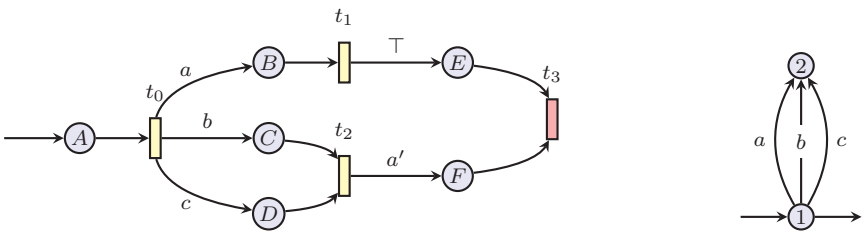
- each pair in the output of the transition can be "validated" by the graph:

$$\begin{aligned} \forall p \in {}^{\triangleright}t_k, \forall \langle x, q\rangle \in {}^{\triangleleft}t_k, \qquad & x \in \Sigma \Rightarrow \langle \rho_k(p), x, \rho_{k+1}(q)\rangle \in E, \\ & x = y' \Rightarrow \langle \rho_{k+1}(q), y, \rho_k(p)\rangle \in E, \\ & x = 1 \Rightarrow \rho_k(p) = \rho_{k+1}(q). \end{aligned}$$

The *language of a run* R, denoted by $\mathscr{L}(R)$, is the set of graphs that can be read along R. ∗

Notice that there is no "validation" required for transitions labelled with $\top$. This means that such a transition allows the reading map to jump to any vertex in the graph.

Example 4.17.
The following automaton (on the left) can read the following graph (on the right):



Let R be the run $\langle \{A\}, t_0; t_1; t_2; t_3, \emptyset\rangle$. To read the graph above along R, one should use the following reading:

$$[A \mapsto 1]; \begin{bmatrix} B \mapsto 2 \\ C \mapsto 2 \\ D \mapsto 2 \end{bmatrix}; \begin{bmatrix} E \mapsto 1 \\ C \mapsto 2 \\ D \mapsto 2 \end{bmatrix}; \begin{bmatrix} E \mapsto 1 \\ F \mapsto 1 \end{bmatrix}; [].$$

One can easily check that this is indeed a valid reading along R. •

The language of a Petri automaton is finally obtained by considering all accepting runs.

Definition 4.18 (Language recognised by a Petri automaton)**.**
The *language recognised by* $\mathscr{A}$, written $\mathscr{L}(\mathscr{A})$, is the following set of graphs:

$$\mathscr{L}(\mathscr{A}) := \bigcup_{R \in \mathrm{Run}_{\mathscr{A}}^{acc}} \mathscr{L}(R)\ .$$ ∗

To avoid confusions between the languages $\mathscr{L}(\mathscr{A})$ and $\mathscr{G}(\mathscr{A})$, we write "G is produced by $\mathscr{A}$" when $G \in \mathscr{G}(\mathscr{A})$, reserving language theoretic terminology like "G is accepted by $\mathscr{A}$" or "$\mathscr{A}$ recognises G" to cases where we mean $G \in \mathscr{L}(\mathscr{A})$.

Lemma 4.19. *For any accepting run R, we have $G \in \mathscr{L}(R)$ if and only if $G \blacktriangleleft \lceil \mathscr{G}(R) \rceil$.* ■

Proof. Let us fix a Petri automaton $\mathscr{A} = \langle P, \mathscr{T}, \iota_{\mathscr{A}} \rangle$. Let $G = \langle V, E, \iota, o \rangle$ and $R = \langle \{\iota\}, t_0; \ldots; t_n, \emptyset \rangle$, with intermediary states $S_1, \ldots$.

Remember that for every k and $p \in \pi_2\left({}^{\triangleleft}t_k\right)$, we defined (Definition 3.7):

$$\nu(k,p) = \{l \mid l > k \text{ and } p \in {}^{\triangleright}t_l\}.$$

$\mathscr{G}(R)$, is then the graph with vertices $\{0, \ldots, n\}$ and the set of edges defined by:

$$E_R = \{\langle k, a, l \rangle \mid \langle a, p \rangle \in {}^{\triangleleft}t_k \text{ and } l = \min \nu(k,p)\}.$$

Finally, we get $\lceil \mathscr{G}(R) \rceil = \langle \{[i] \mid 0 \leqslant i \leqslant j\}, E', [0], [n] \rangle$. (It is quite easy to check that the source of $\mathscr{G}(R)$ is the vertex 0, and that its sink is n.)

It will prove convenient in the following to use the notation $\mathcal{N}(k,p) = \min \nu(k-1,p) = \min\left\{l \;\middle|\; l \geqslant k \text{ and } p \in {}^{\triangleright}t_l\right\}$. Notice that $\forall k, p \in S_k, k \leqslant \mathcal{N}(k,p) \leqslant n$, and that whenever $p \in {}^{\triangleright}t_k$, we have $\mathcal{N}(k,p) = k$.

Suppose there exists a graph homomorphism φ from $\lceil \mathscr{G}(R) \rceil$ to G. We build a reading $(\rho_k)_k$ of G along R by letting $\rho_k(p) := \varphi([\mathcal{N}(k,p)])$ for $0 \leqslant k \leqslant n$ and $p \in S_k$. We now have to check that ρ is truly a reading of G in $\mathscr{A}$:

- for the initialisation of the reading:

$$\begin{aligned} \rho_0(\iota_{\mathscr{A}}) &= \varphi([\mathcal{N}(0, \iota_{\mathscr{A}})]) && \text{(by definition)} \\ &= \varphi([0]) = \{\iota\} && (\varphi \text{ is a homomorphism}) \end{aligned}$$

- for the final transition:

$$\begin{aligned} p \in S_n,\ \rho_n(p) &= \varphi([\mathcal{N}(n,p)]) \\ &= \varphi([n]) = \{o\}. && (\varphi \text{ is a homomorphism}) \end{aligned}$$

- for all $p \in {}^{\triangleright}t_k$, $\rho_k(p) = \varphi([\mathcal{N}(k,p)]) = \varphi([k])$ which does not depend on p.
- for all $p \in S_{k+1} \setminus {}^{\triangleright}t_k$, we have $\mathcal{N}(k,p) = \mathcal{N}(k+1,p)$ (since $p \notin {}^{\triangleright}t_k$). Hence

$$\begin{aligned} \rho_k(p) = \varphi([\mathcal{N}(k,p)]) &= \varphi([\mathcal{N}(k+1,p)]) \\ &= \rho_{k+1}(p). \end{aligned}$$

- for all $p \in {}^{\triangleright}t_k$ and $\langle x, q \rangle \in {}^{\triangleleft}t_k$, we know that $\rho_k(p) = \varphi([k])$ and that $\langle k, x, \mathcal{N}(k+1,q) \rangle \in E_R$.
  - If $x \in \Sigma$, we also have $\langle [k], x, [\mathcal{N}(k+1,q)] \rangle \in E'$. Because φ is a homomorphism we can deduce that:

    $$\langle \varphi([k]), x, \varphi([\mathcal{N}(k+1,q)]) \rangle \in E,$$

    which can be rewritten $\langle \rho_k(p), x, \rho_{k+1}(q) \rangle \in E$.
  - If $x = y'$, $y \in \Sigma$, we also have $\langle [\mathcal{N}(k+1,q)], y, [k] \rangle \in E'$. Because φ is a homomorphism we can get like before $\langle \rho_{k+1}(q), y, \rho_k(p) \rangle \in E$.
  - If finally $x = 1$, then we know that $k \equiv \mathcal{N}(k+1,q)$, thus proving that

    $$\rho_k(p) = \varphi([k]) = \varphi([\mathcal{N}(k+1,q)]) = \rho_{k+1}(q).$$

If on the other hand we have a reading $(\rho_k)_{0\leqslant k\leqslant n}$ of G, we define $\varphi : \{0,\dots,n\} \to V$ by $\varphi([k]) := \rho_k(p)$ for any $p \in {}^{\triangleright}t_k$. As $(\rho_k)_k$ is a reading, φ is well defined[3]. Let us check that φ is a homomorphism from $\lceil \mathscr{G}(R) \rceil$ to G:

- $\varphi([0]) = \rho_0(\iota_{\mathscr{A}}) = \iota$;
- $\varphi([n]) = \rho_n(p)$ with $p \in S_n$, and since $(\rho_k)_k$ is a reading and t_n is final, $\rho_n(p) = o$.
- if $\langle [k], x, [l]\rangle \in E'$ is an edge of $\lceil \mathscr{G}(R) \rceil$, then it was produced from some edge $(i, y, j) \in E_R$, with either $x = y$ and $\langle i, j\rangle \in [k] \times [l]$ or $y = x'$ and $\langle i, j\rangle \in [l] \times [k]$. There is some $p \in {}^{\triangleright}t_i$ and q such that $\langle y, q\rangle \in {}^{\triangleleft}t_j$ and $j = \mathcal{N}(i+1, q)$.

  By definition of $\mathcal{N}$ we know that $\forall i+1 \leqslant m < j, q \notin {}^{\triangleright}t_m$. Thus, because $(\rho_k)_k$ is a reading, $\rho_{i+1}(q) = \rho_j(q)$ and either $\langle \rho_i(p), x, \rho_{i+1}(q)\rangle \in E$ or $\langle \rho_{i+1}(q), x, \rho_i(p)\rangle \in E$, and thus $\langle \varphi([k]), x, \varphi([l])\rangle \in E$. □

As an immediate corollary, we obtain the following characterisation of the language of a Petri automaton.

Corollary 4.20. $\mathscr{L}(\mathscr{A}) = {}^{\blacktriangleleft}\lceil \mathscr{G}(\mathscr{A}) \rceil$. ■

The left-hand side language is defined through readings along accepting runs, which is a local and incremental notion and which allows us to define *simulations* in Section 4.5.3. By contrast, the right-hand side language is defined globally.

Corollary 4.21. *The (in)equational theory of bounded Kleene allegories is co-recursively enumerable.* ■

Proof. Construct Petri automata for the two expressions and enumerate all potential counter-examples, i.e., graphs. A graph is a counter-example if it can be read in one automaton but not in the other, which is a decidable property. □

We also obtain a Kleene Theorem for Bounded Kleene Allegories:

Corollary 4.22. *For every set of graphs G, there is an allegorical expression e such that ${}^{\blacktriangleleft}\mathfrak{G}(e) = G$ if and only if there is a Petri automaton $\mathscr{A}$ with $\mathscr{L}(\mathscr{A}) = G$.* ■

Proof. Let $e \in \mathrm{AReg}\langle\Sigma\rangle$ in converse-normal form (wlog), we have:

$$\begin{aligned} {}^{\blacktriangleleft}\mathfrak{G}(e) &= {}^{\blacktriangleleft}\lceil \mathscr{G}(\mathscr{A}\lfloor e \rfloor) \rceil && \text{(Lemma 4.15)} \\ &= \mathscr{L}(\mathscr{A}\lfloor e \rfloor). && \text{(Corollary 4.20)} \end{aligned}$$

On the other hand, let $\mathscr{A}$ be any Petri automaton over $\Sigma_\bullet$, because of the Kleene theorem we proved in Chapter 3 (Theorem 3.35) we know how to build an expression $e_{\mathscr{A}} \in \mathrm{GReg}\langle\Sigma_\bullet\rangle$ such that:

$$\mathscr{G}(\mathscr{A}) = \mathscr{G}(e_{\mathscr{A}}). \tag{4.6}$$

From that expression, we may build an expression $e'_{\mathscr{A}} \in \mathrm{AReg}\langle\Sigma\rangle$ by replacing every occurrence of a' with $a^\smile$, and simply looking at 1 and $\top$ as constants rather than letters. By design, we get:

$$\lfloor e'_{\mathscr{A}} \rfloor = e_{\mathscr{A}}. \tag{4.7}$$

Hence we obtain:

$$\begin{aligned} \mathscr{L}(\mathscr{A}) &= {}^{\blacktriangleleft}\lceil \mathscr{G}(\mathscr{A}) \rceil && \text{(Corollary 4.20)} \\ &= {}^{\blacktriangleleft}\lceil \mathscr{G}(e_{\mathscr{A}}) \rceil && \text{(Equation (4.6))} \\ &= {}^{\blacktriangleleft}\lceil \mathscr{G}(\lfloor e'_{\mathscr{A}} \rfloor) \rceil && \text{(Equation (4.7))} \\ &= {}^{\blacktriangleleft}\mathfrak{G}(e'_{\mathscr{A}}). && \text{(Proposition 4.14)} \end{aligned}$$

□

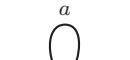
[3] It is not difficult to check that $k \equiv l \Rightarrow \forall \langle p, q\rangle \in {}^{\triangleright}t_k \times {}^{\triangleright}t_l, \rho_k(p) = \rho_l(q)$.

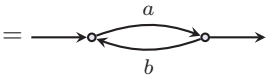
Together with Theorem 4.9, this proves that from a decidability stand point, language equivalence of Petri automata and equivalence of allegorical expressions in relational models are equivalent.

4.5 Comparing automata

4.5.1 Restriction

The above results hold for the whole syntax of regular expressions with converse, intersection and $\top$. However, in the remainder of the chapter, we have to focus on expressions without converse, $\top$, nor identity (what we called graph expressions in the previous chapter). This is because in combination with intersection, the converse and identity operations operations introduce cycles in the graphs associated to ground terms. Consider for instance the graphs for $a \cap 1$ and $a \cap b^\smile$:

$G(a \cap 1) =$ 

$G(a \cap b^\smile) =$ 

.

We thus work with expressions from $\mathrm{GReg}\langle\Sigma\rangle$. Accordingly, ground terms are restricted to the following syntax:

$$u, v, w ::= x \in \Sigma \mid u \cdot v \mid u \cap v.$$

Every expression in $\mathrm{GReg}\langle\Sigma\rangle$ is evidently in converse normal form, and $\lfloor e \rfloor = e$. In particular, the alphabet is still Σ, rather than $\Sigma_\bullet$. Hence we get automata labelled with Σ. For clarity, we call such automata *simple Petri automata* in the following. Furthermore in such an automaton $\mathscr{A}$ for every run $R \in \mathrm{Run}^{acc}_{\mathscr{A}}$ we have $\lceil \mathscr{G}(R) \rceil = \mathscr{G}(R)$. This allows us to obtain:

$$\mathscr{L}(\mathscr{A}) = {}^{\blacktriangleleft}\mathscr{G}(\mathscr{A}). \tag{4.8}$$

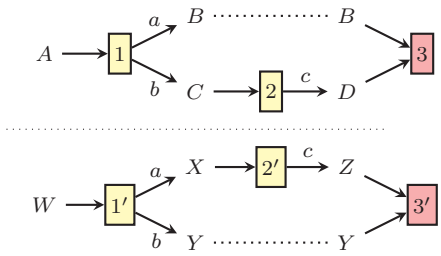
These observations allow us to state the following fact, which will be the basis for the algorithm we present next.

Fact 4.23. *For every pair of expressions $e, f \in \mathrm{GReg}\langle\Sigma\rangle$, the following are equivalent:*

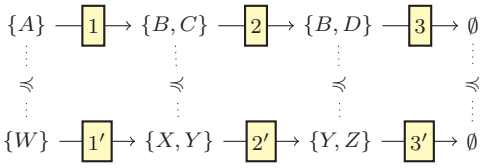
(i) $\mathcal{R}el \models e \leq f$, *(ii)* $\forall R \in Run^{acc}_{\mathscr{A}(e)}, \mathscr{G}(R) \in \mathscr{L}(\mathscr{A}(f))$. ■

4.5.2 Intuitions

In this section, we show how the notion of simulation relation, that allows one to compare NFA, can be adapted to handle simple Petri automata. Consider two automata $\mathscr{A}_1 = \langle P_1, \mathscr{T}_1, \iota_1\rangle$ and $\mathscr{A}_2 = \langle P_2, \mathscr{T}_2, \iota_2\rangle$, we try to show that for any accepting run R in $\mathscr{A}_1$, $\mathscr{G}(R)$ is recognised by some accepting run R' in $\mathscr{A}_2$. Leaving non-determinism aside, the first idea that comes to mind is to find a relation between the states in $\mathscr{A}_1$ and the states in $\mathscr{A}_2$, that satisfy some conditions on the initial and final states, and such that if $S_k \preccurlyeq S'_k$ and $\mathscr{A}_1 \vdash t : S_k \to S_{k+1}$, then there is a state S'_{k+1} in $\mathscr{A}_2$ such that $S_{k+1} \preccurlyeq S'_{k+1}$, $\mathscr{A}_2 \vdash t' : S'_k \to S'_{k+1}$, and these transition steps are compatible in some sense. However, such a definition will not give us the result we are looking for. Consider these two runs:

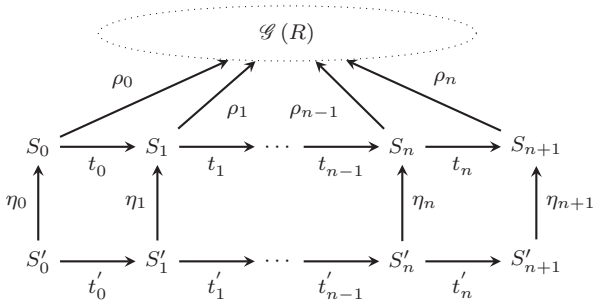


The graphs produced by the first and the second runs correspond respectively to the ground terms $a \cap (b \cdot c)$ and $(a \cdot c) \cap b$. These two terms are incomparable, but the relation $\preccurlyeq$ depicted below satisfies the previously stated conditions.



The problem here is that in Petri automata, runs are token firing games. To adequately compare two runs, we need to closely track the tokens. For this reason, we will relate a state S_k in $\mathscr{A}_1$ not only to a state S'_k in $\mathscr{A}_2$, but to a map η_k from S'_k to S_k. This will enable us to associate with each token situated on some place in P_2 another token placed on $\mathscr{A}_1$.

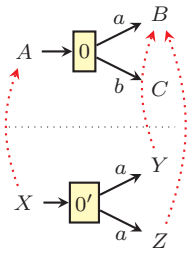
We want to find a reading of $\mathscr{G}(R)$ in $\mathscr{A}_2$, i.e., a run in $\mathscr{A}_2$ together with a sequence of maps associating places in $\mathscr{A}_2$ to vertices in $\mathscr{G}(R)$. Consider the picture below. Since we already have a reading of $\mathscr{G}(R)$ along R (by defining $\rho_k(p) = \mathcal{N}(k,p)$, as in the proof of Lemma 4.19), it suffices to find maps from the places in $\mathscr{A}_2$ to the places in $\mathscr{A}_1$ (the maps η_k): the reading of $\mathscr{G}(R)$ in $\mathscr{A}_2$ will be obtained by composing η_k with ρ_k.



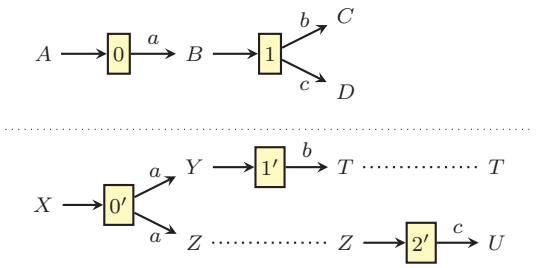
We need to impose some constraints on the maps (η_k) to ensure that $(\rho_k \circ \eta_k)_{0 \leqslant k \leqslant n}$ is indeed a correct reading in $\mathscr{A}_2$. First, we need to ascertain that a transition t'_k in $\mathscr{A}_2$ may be fired from the reading state $\rho_k \circ \eta_k$ to reach the reading state $\rho_{k+1} \circ \eta_{k+1}$. Furthermore, as for NFA, we want transitions t_k and t'_k to be related: specifically, we require t'_k to be included (via the homomorphisms η_k and η_{k+1}) in the transition t_k. This is meaningful because transition t_k contains a lot of information about the vertex k of $\mathscr{G}(R)$ and about ρ: the labels of the outgoing edges from k are the labels on the output of t_k, and the only places that will ever be mapped to k in the reading ρ are exactly the places in the input of t_k.

This already shows an important difference between the simulations for NFA and Petri automata. For NFA, we relate a transition $p \xrightarrow{a} p'$ to a transition $q \xrightarrow{a} q'$ with the same label a. Here the transitions $\mathscr{A}_1 \vdash t_k : S_k \to S_{k+1}$ and $\mathscr{A}_2 \vdash t'_k : S'_k \to S'_{k+1}$ may have different

labels. Consider the step represented below, corresponding to a square in the above diagram. The output of transition 0 has a label b that does not appear in $0'$, and $0'$ has two outputs labelled by a. Nevertheless this satisfies the conditions informally stated above, indeed, $a \cap b \leq a \cap a$ holds.



However this definition is not yet satisfactory. Consider the two runs below:



Their produced graphs correspond respectively to the ground terms $a \cdot (b \cap c)$ and $(a \cdot b) \cap (a \cdot c)$. The problem is that $a \cdot (b \cap c) \leq (a \cdot b) \cap (a \cdot c)$, but with the previous definition, we cannot relate these runs: they do not have the same length. The solution here consists in grouping the transitions $1'$ and $2'$ together, and considering these two steps as a single step in a *parallel run*. This last modification gives us a notion of simulation that suits our needs.

4.5.3 Simulations

Before getting to the notion of simulation, we need to define what is a parallel run, and a parallel reading.

A set of transitions $T \subseteq \mathscr{T}$ is *compatible* if their inputs are pairwise disjoint. If furthermore all transitions in T are enabled in a state S, one can observe that the state S' reached after firing them successively does not depend on the order in which they are fired. In that case we write $\mathscr{A} \vdash T : S \to S'$.

A *parallel run* is a sequence $\mathcal{R} = \langle S_0, T_0; \ldots; T_n, S_{n+1} \rangle$, where the $T_k \subseteq \mathscr{T}$ are compatible sets of transitions such that $\mathscr{A} \vdash T_k : S_k \to S_{k+1}$. We define a *parallel reading* ρ along some parallel run $\mathcal{R} = \langle S_0, T_0; \ldots; T_n, \emptyset \rangle$ by requiring that: $\rho_0(S_0) = \{\iota\}$, $\rho_n(S_n) = \{o\}$, and $\forall k \leqslant n$ the following holds:

- $\forall p \in S_k \setminus \bigcup_{t \in T_k} {}^\triangleright t,\ \rho_{k+1}(p) = \rho_k(p)$;
- $\forall t \in T_k, \forall p, q \in {}^\triangleright t, \rho_k(p) = \rho_k(q)$;
- $\forall t \in T_k, \forall p \in {}^\triangleright t, \forall \langle x, q \rangle \in {}^\triangleleft t$,

$$\begin{aligned} x \in \Sigma &\Rightarrow \langle \rho_k(p), x, \rho_{k+1}(q) \rangle \in E, \\ x = y' \text{ and } y \in \Sigma &\Rightarrow \langle \rho_{k+1}(q), y, \rho_k(p) \rangle \in E, \\ x = 1 &\Rightarrow \rho_k(p) = \rho_{k+1}(q). \end{aligned}$$

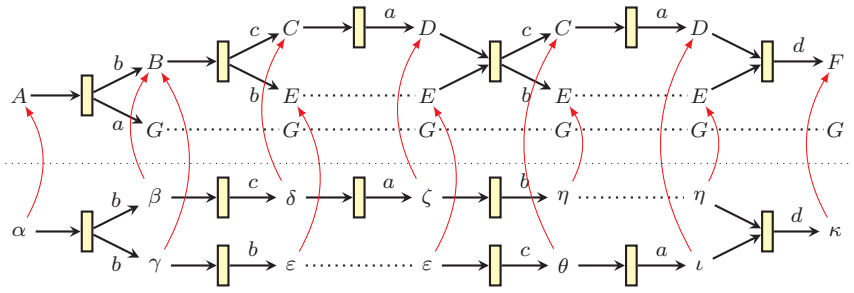


Figure 4.2: Embedding of a parallel run into the run from Figure 3.5.

Remark. We defined parallel readings for arbitrary Petri automata for the sake of generality. However, we will only use them for simple Petri automata in the following.

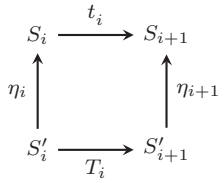
Definition 4.24 (Simulation)**.**
A relation $\preccurlyeq \subseteq \mathcal{P}(P_1) \times \mathcal{P}(P_2 \rightharpoonup P_1)$ between the states of $\mathscr{A}_1$ and the partial maps from the places of $\mathscr{A}_2$ to the places of $\mathscr{A}_1$ is called a *simulation* between $\mathscr{A}_1$ and $\mathscr{A}_2$ if:

- if $S \preccurlyeq E$ and $\eta \in E$ then the range of η must be included in S;
- $\{\iota_1\} \preccurlyeq \{[\iota_2 \mapsto \iota_1]\}$;
- if $S \preccurlyeq E$ and $\mathscr{A}_1 \vdash t : S \to S'$, then $S' \preccurlyeq E'$ where E' is the set of all η' such that there is some $\eta \in E$ and a compatible set of transitions $T \subseteq \mathscr{T}_2$ such that:
  - $\mathscr{A}_2 \vdash T : \mathrm{dom}(\eta) \to \mathrm{dom}(\eta')$;
  - $\forall t' \in T, \eta({}^\triangleright t') \subseteq {}^\triangleright t$ and $\forall \langle x, q\rangle \in {}^\triangleleft t', \langle x, \eta'(q)\rangle \in {}^\triangleleft t$;
  - $\forall p \in \mathrm{dom}(\eta), \left(\forall t' \in T, p \notin {}^\triangleright t'\right) \Rightarrow \eta(p) = \eta'(p)$.
- if $S \preccurlyeq E$ and $S = \emptyset$, then there must be some $\eta \in E$ such that $\mathrm{dom}(\eta) = \emptyset$. ∗

We will now prove that the language of $\mathscr{A}_1$ is contained in the language of $\mathscr{A}_2$ if and only if there exists such a simulation. We first introduce the following notion of embedding.

Definition 4.25 (Embedding)**.**
Let $R = \langle S_0, t_0; \ldots; t_{n-1}, S_n\rangle$ be a run in $\mathscr{A}_1$, and $\mathcal{R} = \langle S'_0, T_0; \ldots; T_{n-1}, S'_n\rangle$ a parallel run in $\mathscr{A}_2$. An *embedding* of $\mathcal{R}$ into R is a sequence $(\eta_i)_{0 \leqslant i \leqslant n}$ of maps such that for any $i < n$, we have:

- η_i is a map from S'_i to S_i;
- the image of T_i by η_i is included in t_i, meaning that for any $t \in T_i$, for any $p \in {}^\triangleright t$ and $\langle x, q\rangle \in {}^\triangleleft t$, $\eta_i(p)$ is contained in the input of t_i and $\langle x, \eta_{i+1}(q)\rangle$ is in the output of t_i;
- the image of the tokens in S_i that do not appear in the input of T_i are preserved ($\eta_i(p) = \eta_{i+1}(p)$) and their image is not in the input of t_i.



∗

Figure 4.2 illustrates the embedding of some parallel run, into the run presented in Figure 3.5. Notice that it is necessary to have a parallel run instead of a simple one: to find something that matches the second transition in the upper run, we need to fire two transitions in parallel in the lower run.

There is a close relationship between simulations and embeddings:

Lemma 4.26. *Let $\mathscr{A}_1$ and $\mathscr{A}_2$ be two Petri automata, the following are equivalent:*

(i) there exists a simulation $\preccurlyeq$ between $\mathscr{A}_1$ and $\mathscr{A}_2$;

(ii) for any accepting run R in $\mathscr{A}_1$, there is an accepting parallel run $\mathcal{R}$ in $\mathscr{A}_2$ that can be embedded into R. ■

Proof. If we have a simulation $\preccurlyeq$, let $R = \langle S_0, t_0; \dots; t_n, \emptyset\rangle$ be an accepting run in $\mathscr{A}_1$. By the definition of simulation, we can find a sequence of sets of maps $(E_k)_{0\leqslant k\leqslant n+1}$ such that $E_0 = \{[\iota_2 \mapsto \iota_1]\}$ and $\forall k, S_k \preccurlyeq E_k$. Furthermore, we can extract from this a sequence of maps $(\eta_k)_{0\leqslant k\leqslant n+1}$ and a sequence of parallel transitions $(T_k)_{0\leqslant k\leqslant n}$ such that:

- $\forall k, \mathscr{A}_2 \vdash T_k : \mathrm{dom}\,(\eta_k) \to \mathrm{dom}\,(\eta_{k+1})$
- the parallel run $\langle \mathrm{dom}\,(\eta_0), T_0; \dots; T_n; \mathrm{dom}\,(\eta_{n+1})\rangle$ is accepting, and can be embedded into R using (η_k).

This follows directly from the definitions of embedding and simulation.

On the other hand, if we have property (ii), then we can define a relation $\preccurlyeq$ by saying that $S \preccurlyeq E$ if there is an accepting run $R = \langle S_0, t_0; \dots; t_n, S_{n+1}\rangle$ in $\mathscr{A}_1$ such that there is an index k_0: $S = S_{k_0}$; and the following holds: $\eta \in E$ if there is an accepting parallel run $\mathcal{R}$ in $\mathscr{A}_2$ visiting successively the states $S'_0, \dots, S'_{n+1}$ and $(\eta'_k)_{0\leqslant k\leqslant n+1}$ an embedding of $\mathcal{R}$ into R such that $\eta = \eta'_{k_0}$. It is then immediate to check that $\preccurlyeq$ is indeed a simulation. □

If η is an embedding of $\mathcal{R}$ into R, and ρ is a reading of $\mathscr{G}(R)$ along R, then we can easily check that $(\rho_i \circ \eta_i)_{0\leqslant i\leqslant n}$ is a parallel reading of $\mathscr{G}(R)$ along $\mathcal{R}$ in $\mathscr{A}_2$. Thus, it is clear that once we have such a run $\mathcal{R}$ with the sequence of maps η, we have that $\mathscr{G}(R)$ is indeed in the language of $\mathscr{A}_2$. The more difficult question is the completeness of this approach: if $\mathscr{G}(R)$ is recognised by $\mathscr{A}_2$, is it always the case that we can find a run $\mathcal{R}$ that may be embedded into R? The answer is affirmative, thanks to Lemma 4.27 below. If $(\rho_j)_{0\leqslant j\leqslant n+1}$ is a reading of G along $R = \langle S_0, t_0; \dots; t_n, S_{n+1}\rangle$, we write $active(j)$ for the only position in $\rho_j({}^{\triangleright}t_j)$[4]. A binary relation $\sqsubseteq$ is a *topological ordering* on $G = \langle V, E, \iota, o\rangle$ if $\langle V, \sqsubseteq\rangle$ is a linear order and $(p, x, q) \in E$ entails $p \sqsubseteq q$.

Lemma 4.27. *Let $G \in \mathscr{L}(\mathscr{A}_2)$ and $\sqsubseteq$ be any topological ordering on G. Then there exists a run R and a reading $(\rho_j)_{0\leqslant j\leqslant n+1}$ of G along R such that $\forall k, active(k) \sqsubseteq active(k+1)$.* ■

The proof of this result is achieved by taking any run R accepting G, and then exchanging transitions in R according to $\sqsubseteq$, while preserving the existence of a reading. However we need to introduce some lemmas first.

Let us fix $\mathscr{A} = \langle P, \mathscr{T}, \iota\rangle$ a Petri automaton, and $R = \langle S_0, t_0; \dots; t_n, S_{n+1}\rangle$ a run of $\mathscr{A}$.

Definition 4.28 (Exchangeable transitions)**.**
Two transitions t_k and t_{k+1} are *exchangeable* in R if for all $p \in S_k$, p is in ${}^{\triangleright}t_{k+1}$ implies that there is no $x \in \Sigma$ such that $(x, p) \in {}^{\triangleleft}t_k$. ∗

As the name might suggests, two exchangeable transitions may be exchanged in a run, and any graph read along the initial run can still be read along the permuted run.

Lemma 4.29. *Suppose t_k and t_{k+1} are exchangeable for some $0 \leqslant k < n$. We write $C' = S_k \setminus {}^{\triangleright}t_{k+1} \cup \pi_2\left({}^{\triangleleft}t_{k+1}\right)$. Then $\mathscr{A} \vdash t_{k+1} : S_k \to C'$ and $\mathscr{A} \vdash t_k : C' \to S_{k+2}$. Furthermore, for any graph G, if $G \in \mathscr{L}(R)$, then $G \in \mathscr{L}(R[k \leftrightarrow k+1])$, where:*

$$R[k \leftrightarrow k+1] := \left\langle S_0, t_0; \cdots; t_{k+1}; t_k; \cdots; t_n, S_n\right\rangle.$$ ■

[4]Recall that if $(\rho_j)_{0\leqslant j\leqslant n+1}$ is a reading along R then for any $p, q \in {}^{\triangleright}t_j$, we have $\rho_j(p) = \rho_j(q)$.

Proof. The fact that $S_k \xrightarrow{t_{k+1}}_{\mathscr{A}} C'$ and $C' \xrightarrow{t_k}_{\mathscr{A}} S_{k+2}$ is trivial to check, with the definition of exchangeable.

Let $(\rho_j)_{0\leqslant j\leqslant n+1}$ be a reading of G along R. If $\left(\rho'_j\right)_{0\leqslant j\leqslant n+1}$ is defined by:

$$\rho'_j(p) = \begin{cases} \rho_j(p) & \text{if } j \neq k+1, \\ \rho_{k+2}(p) & \text{if } j = k+1 \text{ and } (x,p) \in {}^{\triangleleft}t_{k+1} \text{ for some } x, \\ \rho_k(p) & \text{otherwise.} \end{cases}$$

Then $\left(\rho'_j\right)_{0\leqslant j\leqslant n+1}$ is a reading of G along $R[k \leftrightarrow k+1]$. □

Recall that if $(\rho_j)_{0\leqslant j\leqslant n}$ is a reading of G along ξ we write $active(j)$ for the only position in $\rho_j({}^{\triangleright}t_j)$.

Lemma 4.30. *Let $\sqsubseteq$ be any topological ordering on G. If $(\rho_j)_{0\leqslant j\leqslant n+1}$ is a reading of G along R, and if $active(k+1) \sqsubseteq active(k)$ for some k, then t_k and t_{k+1} are exchangeable.* ■

Proof. Let $G = \langle V, E, \iota, o\rangle$. As $(\rho_i)_{0\leqslant j\leqslant n+1}$ is a reading, for any $\langle x, p\rangle \in {}^{\triangleleft}t_k$, $\langle active(k), x, \rho_{k+1}(p)\rangle \in E$, thus

$$active(k) \sqsubset \rho_{k+1}(p).$$

We know that $active(k+1) \sqsubseteq active(k)$, meaning by transitivity that $active(k+1) \sqsubset \rho_{k+1}(p)$. Hence

$$active(k+1) \neq \rho_{k+1}(p)$$

and because $(\rho_i)_{0\leqslant j\leqslant n+1}$ is a reading we can infer that $p \notin {}^{\triangleright}t_{k+1}$, thus proving that t_k and t_{k+1} are exchangeable. □

Proof (Proof of Lemma 4.27). Because G is in $\mathscr{L}(\mathscr{A})$, we can find a reading ρ' along some run R. If that reading is not in the correct order, then by Lemma 4.30 we can exchange two transitions and Lemma 4.29 ensures that we can find a corresponding reading. We repeat this process until we get a reading in the correct order. □

(Notice that if G contains cycles, this lemma cannot apply because of the lack of a topological ordering.)

Lemma 4.27 enables us to build an embedding from any reading of $\mathscr{G}(R)$ in $\mathscr{A}_2$.

Lemma 4.31. *Let R a accepting run of $\mathscr{A}_1$. Then $\mathscr{G}(R)$ is in $\mathscr{L}(\mathscr{A}_2)$ if and only if there is an accepting parallel run in $\mathscr{A}_2$ that can be embedded into R.* ■

Proof. Let $R = \langle S_0, t_0; \ldots; t_n, S_{n+1}\rangle$ be a run. For all indexes k and places $p \in S_k$, define $\rho^1_k(p) := \mathcal{N}(k,p) = \min\left\{l \;\middle|\; l \geqslant k \text{ and } p \in {}^{\triangleright}t_l\right\}$. We have that $\left(\rho^1_k\right)_{0\leqslant k\leqslant n+1}$ is a reading of $\mathscr{G}(R)$ along R.

- Assume an embedding $(\eta_k)_{0\leqslant k\leqslant n+1}$ of an accepting parallel run $\mathcal{R}$ into R. We define a parallel reading (ρ^2_k) of $\mathscr{G}(R)$ in $\mathscr{A}_2$ by letting $\rho^2_k(p) := \rho^1_k(\eta_k(p))$.
- On the other hand, notice that the natural ordering on $\mathbb{N}$ is a topological ordering on $\mathscr{G}(R)$, and that $\forall 0 \leqslant k \leqslant n$, $\rho^1_k({}^{\triangleright}t_k) = \{k\}$. By Lemma 4.27 we gather that $\mathscr{G}(R)$ is in $\mathscr{L}(\mathscr{A}_2)$ if and only if there exists a reading $(\rho^2_j)_{0\leqslant j\leqslant n'}$ of $\mathscr{G}(R)$ along some run $R' = \left\langle S'_0, t'_0; \ldots; t'_{n'}, S'_{n'+1}\right\rangle$ such that $\forall j, active(j) \leqslant active(j+1)$ (with $active(j)$ the only position in $\rho^2_j\left({}^{\triangleright}t_j\right)$).

  Now, suppose we have such a reading; we can build an embedding $(\eta_k)_{0\leqslant k\leqslant n+1}$ as follows. For $k \leqslant n$, define $T_k := \left\{t'_j \;\middle|\; active(j) = k\right\}$. We describe the construction incrementally:

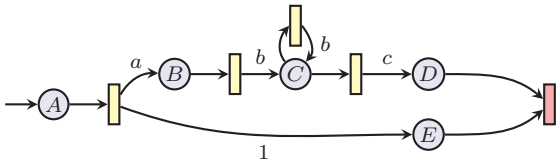


Figure 4.3: A Petri automaton for $1 \cap a \cdot b^+ \cdot c$.

- $\eta_0 = [\iota_2 \mapsto \iota_1]$.
- For all $p \in \mathrm{dom}\,(\eta_k) \setminus \bigcup_{t \in T_k} {}^{\triangleright}t$ we simply set $\eta_k(p) = \eta_{k-1}(p)$.
- Otherwise, $\forall t_j^{'} \in T_k$, let $q \in {}^{\triangleright}t_j^{'}$. Then, for all $\langle x, p\rangle$ in ${}^{\triangleleft}t_j^{'}$, because ρ^2 is a reading and by construction of $\mathscr{G}(R)$ we also know that there is some $p' \in S_{k+1}$ that satisfies $\langle x, p'\rangle \in {}^{\triangleleft}t_k$ and $\rho^1_{k+1}(p') = \rho^2_{j+1}(p)$. That p' is a good choice for p, hence we define $\eta_k(p) = p'$.

It is then administrative to check that $(\eta_k)_{0 \leqslant k \leqslant n+1}$ is indeed an embedding.

For the if direction, we build a parallel reading from the embedding, as explained above. For the other direction, we consider a reading of $\mathscr{G}(R)$ in $\mathscr{A}_2$ along some run R'. Notice that the natural ordering on $\mathbb{N}$ is a topological ordering on $\mathscr{G}(R)$; we may thus change the order of the transitions in R' (using Lemma 4.27) and group them adequately to obtain a parallel reading $\mathcal{R}$ that embeds in R. □

So we know that the existence of embeddings is equivalent to the inclusion of languages, and we previously established that it is also equivalent to the existence of a simulation relation. Hence, the following characterisation holds:

Proposition 4.32. *Let $\mathscr{A}_1$ and $\mathscr{A}_2$ be two simple Petri automata. $\mathscr{L}(\mathscr{A}_1) \subseteq \mathscr{L}(\mathscr{A}_2)$ if and only if there exists a simulation relation $\preccurlyeq$ between $\mathscr{A}_1$ and $\mathscr{A}_2$.* ■

Proof. By Lemmas 4.19, 4.26 and 4.31. □

As Petri automata are finite, there are finitely many relations in $\mathcal{P}(\mathcal{P}(P_1) \times \mathcal{P}(P_2 \rightharpoonup P_1))$. The existence of a simulation thus is decidable, allowing us to prove the main result:

Theorem 4.33. *Given two expressions $e, f \in GReg\langle\Sigma\rangle$, testing whether $\mathcal{R}el \models e = f$ is decidable.* ■

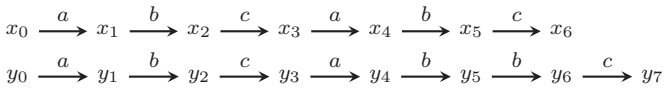
Proof. By Proposition 4.32, Theorems 4.9 and 3.17, and reasoning by double inclusion. □

In practice, we can build the simulation on-the-fly, starting from the pair $\langle\{\iota_1\}, \{[\iota_2 \mapsto \iota_1]\}\rangle$ and progressing from there. We have implemented this algorithm in OCaml [13]. Even though its theoretical worst case time complexity is huge[5], we get a result almost instantaneously on simple one-line examples.

4.5.4 The problems with converse and unit

The previous algorithm is not complete in presence of converse, unit or $\top$. More precisely, Lemma 4.31 does not hold for general automata. Indeed, it is not possible to compare two runs just by relating the tokens at each step, and checking each transition independently. Consider the automaton from Figure 4.3. This automaton has in particular an accepting run recognising $1 \cap abc$. Let us try to test if this is smaller than the following runs from another automaton (we represent the transitions simply as arrows, because they only have a single input and a single output):

[5] A quick analysis gives a $\mathcal{O}\left(2^{n+(n+1)^m}\right)$ complexity bound, where n and m are the numbers of places of the automata.



It stands to reason that we would reach a point where:

- for the first run: $\{D, E\} \preccurlyeq \{[x_3 \mapsto D]\}$;
- for the second run: $\{D, E\} \preccurlyeq \{[y_3 \mapsto D]\}$.

So if it were possible to relate the end of the runs just with this information, they should both be bigger than $1 \cap abc$ or both smaller or incomparable. But in fact the first run (recognising $abcabc$) is bigger than $1 \cap abc$ but the second (recognising $abcabbc$) is not. This highlights the need for having some memory of previously fired transition when trying to compare runs of general Petri automata, thus preventing our local approach to bear fruits. The same kind of example could be found with the converse operation instead of 1.

4.6 Complexity

The previous notion of simulation actually allows us to decide language inclusion of simple automata in ExpSpace. We now show that this problem is in fact ExpSpace-complete.

Lemma 4.34. *Comparing simple Petri automata is* ExpSpace*-easy.* ■

Proof. Our measure for the size of an automaton here is its number of places (the number of transitions is at most exponential in this number). Here is a non-deterministic semi-algorithm that tries to refute the existence of a simulation relation between $\mathscr{A}_1$ and $\mathscr{A}_2$.

1: start with $S := \{\iota_1\}$ and $E := \{[\iota_2 \mapsto \iota_1]\}$;

2: if $S = \emptyset$, check if there is some $\eta \in E$ such that $\mathrm{dom}\,(\eta) = \emptyset$, if not return `FALSE`;

3: choose non-deterministically a transition $t \in \mathscr{T}_1$ such that ${}^\triangleright t \subseteq S$;

4: fire t, which means that $S := S \setminus {}^\triangleright t \cup \pi_2\,({}^\triangleleft t)$;

5: have E progress along t as well, according to the conditions from Definition 4.24.

6: go to step 2.

All these computations can be done in exponential space. In particular as S is a set of places in P_1, it can be stored in space $|P_1| \times \log(|P_1|)$. Similarly, E, being a set of partial functions from P_2 to P_1, each of which of size $|P_2| \times \log(|P_1| + 1)$, can be stored in space $|P_1 + 1|^{|P_2|} \times |P_2| \times \log(|P_1| + 1)$. This non-deterministic ExpSpace semi-algorithm can then be turned into an ExpSpace algorithm by Savitch' theorem [59]. □

One can check that the number of places in $\mathscr{A}(e)$ is linear in the size of e. (The exponential upper-bound on the number of transitions is asymptotically reached, consider for instance the automaton for $(x_1 \cup y_1) \cap (x_2 \cup y_2) \cap \cdots \cap (x_n \cup y_n)$.) Therefore, the previous Lemma gives us a ExpSpace algorithm for deciding the (in)equational theory of identity-free relational Kleene lattices.

To get ExpSpace-hardness, we perform a reduction from the problem RSQ(X): deciding whether a regular expressions with squaring e ($e^2 := e \cdot e$) denotes the universal language $X^\star$. This problem was shown in [46] to be ExpSpace-complete. The proof will follow the method used in [34] to obtain a complexity lower bound on the trace equivalence of finite nets without hidden transitions. To avoid confusion, the regular language denoted by the

expression e will be written $(\!|e|\!)$. Any word u can be seen uniquely as a linear graph $\lambda(u)$. By extension, the set of graphs of words from $(\!|e|\!)$ will be denoted by $\lambda(\!|e|\!)$.

First, notice that if u and v are just words over X, $\lambda(u) \blacktriangleleft \lambda(v)$ is equivalent to $u = v$. Because of this, it is straightforward to check that for any regular expressions with squaring e, f the following holds

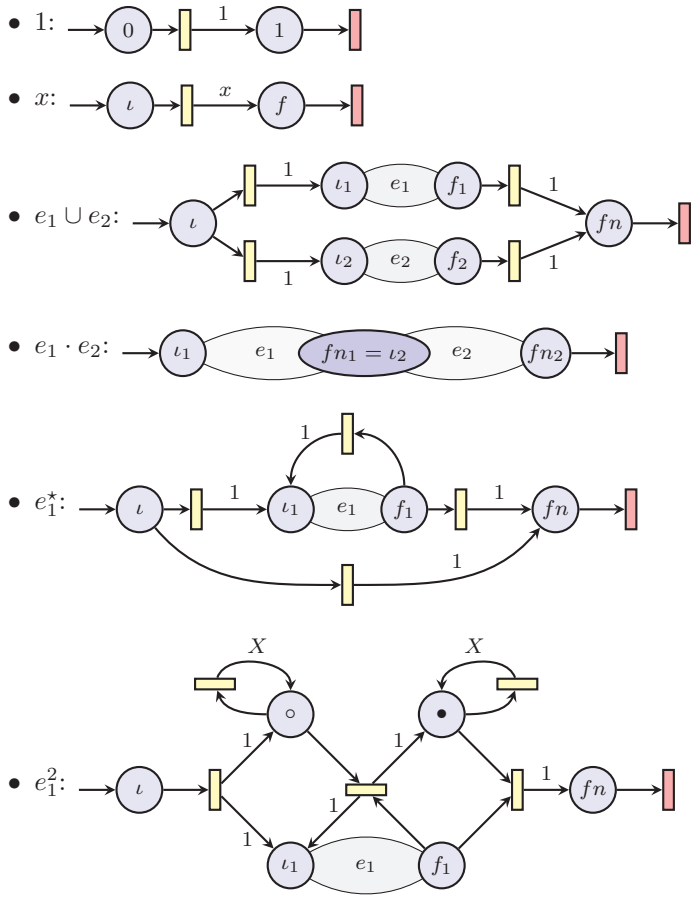
$$^{\blacktriangleleft}(\lambda(\!|e|\!)) = {}^{\blacktriangleleft}(\lambda(\!|f|\!)) \Leftrightarrow (\!|e|\!) = (\!|f|\!). \tag{4.9}$$

We first reduce to the containment of Petri automata with 1.

Lemma 4.35. *The problem of deciding whether the language of a regular expression with squaring is $X^\star$ is polynomial-time reducible to the containment of languages recognised by Petri automata.* ■

Proof. Given an expression e on this signature, we can build in linear time a Petri automaton $\mathscr{A}$, with a linear number of places and transitions. The closure of the language denoted by e is be exactly the language recognised by $\mathscr{A}$.

The automata we produce here only have one final transition, and this transition has a single input. The construction is a straightforward adaptation of Thompson's algorithm for NFA [60]. We describe it graphically.



The only interesting case is for computing an automaton for $e = (e_1)^2$. The transitions labelled by X are a shorthand for a set of transitions, containing for each letter x in X a transition with one output, labelled by x. This construction is linear: the automaton for e_1 is not copied. Furthermore, a run in this automaton will start by sending one token in $\circ$ and one in ι_1, the initial state of the automaton for e_1. Then it will perform a run in this

automaton until a single token reaches the final state for e_1, f_1. At this point the tokens from $\circ$ and f_1 will be sent to $\bullet$ and ι_1, starting a new run of e_1. When a token is finally sent to f_1, it can be consumed together with the one in $\bullet$, to reach the final state.

The automaton for e has at most $4 \times |e|$ places, and at most $(2 \times |X| + 4) \times |e|$ transitions. It is quite obvious that this construction can be performed in linear time and space. By further analysing this construction, one can prove that

1. $\lambda(\!|e|\!) \subseteq \mathscr{L}(\mathscr{A}(e))$;
2. for every $G \in \mathscr{G}(\mathscr{A}(e))$, there is a word $w \in (\!|e|\!)$ such that $G \blacktriangleleft \lambda(w)$.

By combining Items 1 and 2 we establish that

$$\mathscr{L}(\mathscr{A}(e)) = {}^{\blacktriangleleft}\lambda(\!|e|\!).$$

Thus by Equation (4.5), we get that $(\!|e|\!) = (\!|f|\!)$ is equivalent to $\mathscr{L}(\mathscr{A}(e)) = \mathscr{L}(\mathscr{A}(f))$. Hence $\mathscr{L}(\mathscr{A}(X^\star)) \subseteq \mathscr{L}(\mathscr{A}(e))$ is equivalent to $(\!|e|\!) = X^\star$. □

The previous automata are not simple, as they use 1 as a label. We can now get rid of this label, to obtain ExpSpace-hardness for simple Petri automata.

Lemma 4.36. *The problem of deciding whether the language of a regular expression with squaring is $X^\star$ is polynomial-time reducible to the containment of languages recognised by simple Petri automata.* ■

Proof. We use a similar trick as in [34]: we reuse the previous automaton $\mathscr{A}(e)$ by considering it as a simple automaton over the alphabet $X \cup \{1\}$ (thus forgetting about the special semantics of 1, and seeing it simply as a standard letter). Notice that $w \in (\!|e|\!)$ is equivalent to the existence of a word $u \in (X \cup 1)^\star$ such that $\lambda(u) \in \mathscr{L}(\mathscr{A}(e))$ and w can be obtained by erasing from u the occurences of the letter 1. By carefully analysing the automaton $\mathscr{A}(e)$, we may actually impose that u does not contain more that $n_e := (2 \times |X| + 4) \times |e|$ consecutive 1. By adding to each place of $\mathscr{A}(e)$ a transition looping on that place and labelled by 1, one obtains an automaton $\mathscr{A}'(e)$ such that $w \in (\!|e|\!)$ iff the the word $w' \in (1^{n_e}X)^\star 1^{n_e}$ obtained from w by inserting 1s as necessary is accepted by $\mathscr{A}'(e)$. As a consequence $(\!|e|\!) = X^\star$ iff $\mathscr{L}(\mathscr{B}) \subseteq \mathscr{L}(\mathscr{A}'(e))$, where $\mathscr{B}$ is a simple Petri automaton recognising the regular language $(1^{n_e}X)^\star 1^{n_e}$ (there are automata of linear size for this language). □

By combining Lemma 4.34 and Lemma 4.36 we get:

Theorem 4.37. *Comparing simple Petri automata is* ExpSpace*-complete.* ■

This proof does not allow us to deduce that also the (in)equational theory of Kleene allegories is ExpSpace-hard: the Petri automata we construct are not associated to some expressions of polynomial size, a priori.

4.7 Relationship with standard Petri net notions

Our notion of Petri automaton is really close to the standard notion of labelled (safe) Petri net, where the transitions themselves are labelled, rather than their outputs. We motivate this design choice, and we relate some of the notions we introduced to the standard ones [49].

Any Petri automaton can be translated into a safe Petri net whose transitions are labelled by $\Sigma \uplus \{\tau\}$, the additional label τ standing for silent actions. For each automaton transition $\langle \{p_1, \ldots, p_n\}, \{\langle x_1, q_1\rangle, \ldots, \langle x_m, q_m\rangle\}\rangle$ with $m > 1$, we introduce m fresh places $r_1, \ldots, r_m$ and $m+1$ transitions: a silent transition t_0 with preset $\{p_1, \ldots, p_n\}$ and postset $\{r_1, \ldots, r_m\}$; and for each $1 \leqslant k \leqslant m$ a transition t_k labelled by x_k, with preset $\{r_k\}$ and postset $\{q_k\}$.

The inductive construction from Section 3.2.4 is actually simpler to write using labelled Petri nets, as one can freely use τ-labelled transitions to assemble automata into larger ones, one does not need to perform the τ-elimination steps on the fly.

On the other side, we could not define an appropriate notion of simulation for Petri nets: we need to fire several transitions at once in the small net, to provide enough information for the larger net to answer; delimiting which transitions to group and which to separate is non-trivial; similarly, defining a notion of parallel step is delicate in presence of τ-transitions. By switching to our notion of Petri automata, we impose strong constraints about how those τ-transitions should be used, resulting in a more fitted model.

To describe a run in a Petri net N, one may use a *process* $p : K \to N$, where K is an *occurrence net* (a partially ordered Petri net) [31]. The graphical representation (Figures 3.5 and 4.2) we used to describe runs in an automaton are in fact a mere adaptation of this notion to our setting (with labels on arcs rather than on transitions).

Our notion $\mathscr{G}(R)$ of trace of a run actually corresponds to the standard notion of *pomset-trace*[34], via dualisation. Let R be a run in a Petri automaton, and let R' be the corresponding run in the corresponding labelled Petri net. Let $\mathscr{G}(R) = \langle V, E, \iota, \omega \rangle$. It is not difficult to check that the pomset-trace of R' is isomorphic to $\langle E, <_E \rangle$, where $<_E$ is the transitive closure of the relation $<$ defined on E by $\forall x, y, z \in V,\ a, b \in X, \langle x, a, y \rangle < \langle y, b, z \rangle$.

The correspondence is even stronger: two graphs produced by accepting runs in (possibly different) automata $\mathscr{G}(R_1) = \langle V_1, E_1, \iota_1, o_1 \rangle$ and $\mathscr{G}(R_2) = \langle V_2, E_2, \iota_2, o_2 \rangle$ are isomorphic if and only if their pomset-traces $(E_1, <_{E_1})$ and $(E_2, <_{E_2})$ are isomorphic. The proof of this relies on the fact that accepting runs produce graphs satisfying the following properties:

$$\forall \langle x, a, y \rangle \in E,\ x \neq \iota \Leftrightarrow (\exists \langle z, b \rangle \in V \times X : \langle z, b, x \rangle \in E) \quad ,$$
$$\forall \langle x, a, y \rangle \in E,\ y \neq o \Leftrightarrow (\exists \langle z, b \rangle \in V \times X : \langle y, b, z \rangle \in E) \quad .$$

Hence, pomset-trace language equivalence corresponds exactly to equivalence of the sets of produced graphs in our setting (up to isomorphism).

Jategaonkar and Meyer showed that the pomset-trace equivalence problem for safe Petri nets is ExpSpace-complete [34]. However this equivalence is too strong and does not coincide with the one discussed in the present paper, even for simple Petri automata. Pomset-trace equivalence for Petri nets corresponds to equivalence of the sets of graphs produced by Petri automata ($\mathscr{G}(\mathscr{A}) = \mathscr{G}(\mathscr{B})$, up to graph isomorphism). However, for the equational theory we consider, we need to compare the languages, which are downward-closed sets of graphs, (${}^{\blacktriangleleft}\mathscr{G}(\mathscr{A}) = {}^{\blacktriangleleft}\mathscr{G}(\mathscr{B})$, i.e., $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\mathscr{B})$) rather than the sets of graphs themselves.

Also note that the class of sets of traces of Petri automata ($\{\mathscr{G}(\mathscr{A}) \mid \mathscr{A}$ a Petri automaton$\}$) is not downward closed. Intuitively, the width of any graph in $\mathscr{G}(\mathscr{A})$ is bounded by the number number of places of $\mathscr{A}$, but ${}^{\blacktriangleleft}\mathscr{G}(\mathscr{A})$ usually contains graphs of arbitrary width. As a consequence, one cannot easily reduce our problem to pomset-trace equivalence of safe Petri nets.

Fifth chapter

A formal exploration of Nominal Kleene Algebra

"What's in a name? That which we call a rose by any other name would smell as sweet."

— William Shakespeare, *Romeo and Juliet.*

5.1 Introduction

Gabbay and Ciancia introduced a nominal extension of Kleene algebra [28], as a framework for trace semantics with dynamic allocation of resources. The associated semantics extends formal languages into nominal languages, where words have a nominal structure. Kozen et al. recently proved the completeness of the proposed axiomatisation [40], and proposed a coalgebraic treatment [39] yielding decidability of the equational theory.

They use the following syntax for nominal regular expressions:

$$e, f ::= a \in \Sigma \mid 0 \mid 1 \mid e + f \mid e \cdot f \mid e^\star \mid \nu a.e \ ,$$

where Σ is the alphabet, and $\nu a.e$ makes it possible to generate a fresh letter (or name) a before continuing as e. For instance, the expression $\nu a.\nu b.(a{\cdot}b)$ denotes the language of all words of length two consisting of two distinct letters.

While such a syntax is natural from a nominal point of view, other choices are possible. For instance, one might expect expressions to be typed or classified according to their set of free names. Similarly, name permutations, which are available in any model, can be reified at the syntactic level. We first list four axiomatisations of the corresponding presentations—one of them corresponding to Gabbay and Ciancia's axiomatisation, and we prove that all choices are in fact equivalent.

Recall from Section 1.3 that Kleene algebra are complete not only with respect to language models, but also relational models. There, the letters from the alphabet are interpreted as binary relations, and the regular operations correspond to standard operations on binary relations: union, composition, reflexive transitive closure. This makes it possible to interpret imperative programs, seen as state transformers: binary relations between memory states. Kozen actually designed an extension of Kleene algebra, Kleene algebra with tests [37], which makes it possible to represent not only the control flow of such programs, but also the tests and conditions appearing in while loops and branching statements.

Extending Kleene algebra with names seems appropriate to model imperative programs with local variables, where part of the memory is visible only locally. The previous notion of nominal Kleene algebra is however not really appropriate for this purpose: letters of the alphabet (i.e., atomic programs, instructions) are equated with names bound by the $\nu a.e$ construct (i.e., memory locations). In contrast, the instruction $x \leftarrow y$ that assigns to variable x the value of variable y should be an elementary construction depending on the names x and y. For this reason, we provide an extension of the syntax where letters are chosen from an arbitrary nominal set. The typed version of this extension is more appropriate for modelling imperative programs with local variables; like above for plain nominal Kleene algebra, we show that the various presentations are equivalent. We moreover show that the extension is conservative: plain nominal Kleene algebras can be encoded into the extended ones. Whether a converse encoding is possible remains open.

Outline. We define the various theories in Section 5.2 and we compare them in Section 5.3. In Section 5.4 we provide a relational interpretation for our extended model. We conclude in Section 5.5.

5.2 Expressions and proofs

5.2.1 Atoms and letters

Let $\mathcal{A}$ be an infinite set of *atoms* with decidable equality. We consider in this paper *finitely supported permutations of atoms*, simply called permutations in the following. They are bijections π such that there is a finite set $A \subseteq \mathcal{A}$ such that $a \notin A \Rightarrow \pi(a) = a$. The *inverse* of a permutation π is written π^{-1}. The *identity permutation* is denoted by $()$, and the permutation exchanging a and b, and leaving every other atom unchanged, is written $(a\ b)$. Finally, if π is a permutation and A is a finite set of atoms, $\pi(A) := \{\pi(a) \mid a \in A\}$ is *the image of A under π*.

We consider as *letters* an arbitrary nominal set $\mathcal{L}$ [29, 51], which we assume to be decidable. Such a set is specified through the data of its set of elements, a function $\natural() : \mathcal{L} \to \mathcal{P}_f(\mathcal{A})$ mapping every element to its *support*, and an *action* of the group of permutations on $\mathcal{L}$. These functions must satisfy the following axioms:

$$\forall x \in \mathcal{L}, \forall \pi, (\forall a \in \natural(x), \pi(a) = a) \Rightarrow \pi(x) = x. \tag{5.1}$$
$$\forall x \in \mathcal{L}, \forall \pi, \natural(\pi(x)) = \pi(\natural(x)). \tag{5.2}$$
$$\forall x \in \mathcal{L}, \forall \pi, \pi', \pi(\pi'(x)) = \pi \circ \pi'(x). \tag{5.3}$$

5.2.2 Expressions and sets of expressions

We define a single type for expressions, containing all possible operators, and we define several fragments of it afterwards. Doing so makes it possible to share several definitions, enabling important code-reuse in our proof scripts.

Definition 5.1 (Nominal expressions)**.**
The set $\mathbb{E}$ of *expressions* is composed of terms formed over the following syntax, where the letter A is a finite set of atoms, π denotes a permutation, a is an atom and l is a letter:

$$e, f := \ 0 \mid 1 \mid e + f \mid e \cdot f \mid e^\star \mid \nu_a.e \mid l \mid a \mid \langle\pi\rangle\, e \mid \bot_A \mid id_A \mid w_a.e.$$

∗

Product $(\cdot)$, sum $(+)$ and Kleene star $(\cdot^\star)$ are the regular operations, together with the associated constants 0 and 1, ν_a is name restriction. Variables can be either letters l or atoms a. We include a syntactic construction for explicit permutations $\langle\pi\rangle$. The remaining entries ($\bot_A$, id_A, and w_a) are for the presheaf presentation; we will discuss them in a moment.

Untyped expressions

Definition 5.2 (Untyped expression)**.**
An expression e is *untyped* if it neither contains the operator w_a nor the constants $\bot_A$ and id_A. The set of untyped expressions is written $\mathbb{U}$. ∗

We define freshness only for untyped expressions:

Definition 5.3 (Freshness)**.**
An atom a *is fresh for* e if the judgement $a \# e$ can be inferred in the following system.

$$\frac{}{a \# 1} \qquad \frac{}{a \# 0} \qquad \frac{a \notin \natural(l)}{a \# l} \qquad \frac{a \neq b}{a \# b} \qquad \frac{a \# e \quad a \# f}{a \# e + f}$$

$$\frac{a \# e \quad a \# f}{a \# e \cdot f} \qquad \frac{a \# e}{a \# e^\star} \qquad \frac{}{a \# \nu_a.e} \qquad \frac{a \# e}{a \# \nu_b.e} \qquad \frac{\pi^{-1}(a) \# e}{a \# \langle \pi \rangle e}\ .$$

∗

Accordingly, the *support* of an untyped expression e, written $\natural(e)$, is the unique set such that $\forall a,\ a \# e \Leftrightarrow a \notin \natural(e)$.

Typed expressions For the presheaf approach, we replace freshness assumptions with type derivations. In order to enforce uniqueness of types, we replace the constants 0 and 1 from the untyped syntax by the annotated constants $\bot_A$ and id_A, and we use explicit weakenings (w_a).

Definition 5.4 (Typed expressions)**.**
For any $e \in E$ and $A \in \mathcal{P}_f(\mathcal{A})$, e *has the type* A if the judgement $e : A$ can be inferred in the following system:

$$\frac{}{id_A : A} \qquad \frac{}{\bot_A : A} \qquad \frac{l \in \mathcal{L}}{l : \natural(l)} \qquad \frac{a \in \mathcal{A}}{a : \{a\}} \qquad \frac{e : A \quad f : A}{e + f : A}$$

$$\frac{e : A \quad f : A}{e \cdot f : A} \qquad \frac{e : A}{e^\star : A} \qquad \frac{e : A \cup \{a\} \quad a \notin A}{\nu_a.e : A} \qquad \frac{e : A \setminus \{a\} \quad a \in A}{w_a.e : A} \qquad \frac{e : \pi^{-1}(A)}{\langle \pi \rangle e : A}$$

If this is the case, then e is *typed*. The set of typed expressions is written $\mathbb{T}$. ∗

Remark. This type system is syntax directed and yields a simple decision procedure.

Expressions over letters or atoms A significant motivation for this work was to study the differences between having atoms or letters as variables in expressions. Hence we define two other subsets.

Definition 5.5 (Atomic expressions, literate expressions)**.**
An expression e is called *atomic* (respectively *literate*) if it does not contain letters (respectively atoms) as variables. The set of atomic expressions is $\mathbb{E}_\mathcal{A}$, and the set of literate expressions is $\mathbb{E}_\mathcal{L}$. ∗

Intuitively, there are two main differences between having atoms or letters as variables. First, a letter may depend on many atoms. Second, two letters with the same support can still be different, whereas the following equivalence holds :

$$\forall a, b \in \mathcal{A},\ a = b \Leftrightarrow (\forall c \in \mathcal{A},\ c \# a \Leftrightarrow c \# b)\,.$$

Positive expressions For the sake of proofs, we also define the classes of expressions without 0 or $\bot_A$ as a sub-expression. A motivation for excluding these is that in any reasonable system $0 \equiv 0 \cdot e$. Hence if there is an atom a not fresh for e, we would have two equivalent expressions with different sets of fresh variables.

Definition 5.6 (Positive expression)**.**
An expression e is *positive* if it does not contain 0 nor $\bot_A$ as a sub-expression. The set of positive expressions is $\mathbb{E}^+$. ∗

For concision, we write $\mathbb{E}^+_\mathcal{L}$ for $\mathbb{E}_\mathcal{L} \cap \mathbb{E}^+$, and $\mathbb{E}^+_\mathcal{A}$ for $\mathbb{E}_\mathcal{A} \cap \mathbb{E}^+$.

Explicit permutations Our syntax includes for explicit permutations $\langle\pi\rangle$, while permutations are usually considered as external operations. This allows one to manipulate permutations inside the expressions, and we shall see that this addition does not raise the complexity of the problem.

Nevertheless, we need to formally define the semantics of permutations on expressions.

Definition 5.7 (Action of a permutation on an expression)**.**
Let $e \in \mathbb{E}$ be an expression and π a permutation. The *action of* π *on* e, written $\pi \bowtie e$, is defined as follows:

$$\pi \bowtie 1 := 1 \qquad \pi \bowtie 0 := 0 \qquad \pi \bowtie id_A := id_{\pi(A)} \qquad \pi \bowtie \bot_A := \bot_{\pi(A)} \qquad \pi \bowtie a := \pi\,(a)$$

$$\pi \bowtie l := \pi\,(l) \qquad \pi \bowtie (e^\star) := (\pi \bowtie e)^\star \qquad \pi \bowtie (e \cdot f) := \pi \bowtie e \cdot \pi \bowtie f$$

$$\pi \bowtie (e + f) := \pi \bowtie e + \pi \bowtie f \qquad \pi \bowtie (w_a.e) := w_{\pi(a)}.\,(\pi \bowtie e)$$

$$\pi \bowtie (\nu_a.e) := \nu_{\pi(a)}.\,(\pi \bowtie e) \qquad \pi \bowtie (\langle\pi'\rangle\, e) := \langle()\rangle\,(\pi \circ \pi') \bowtie e$$

∗

Expressions without explicit substitutions are called *clean*.

Definition 5.8 (Clean expressions)**.**
An expression e is *clean* if it never uses the operator $\langle\pi\rangle$. The set of clean expressions is $\mathbb{C}$. ∗

Applying permutations preserves all classes we have listed so far:

Lemma 5.9. *For any subset of expressions S chosen from $\{\mathbb{T}, \mathbb{U}, \mathbb{E}_{\mathcal{A}}, \mathbb{E}_{\mathcal{L}}, \mathbb{E}^+, \mathbb{C}\}$, for any permutation π, and for any expression $e \in \mathbb{E}$, $e \in S \Leftrightarrow \pi \bowtie e \in S$. Furthermore, if e has the type A then $\pi \bowtie e : \pi\,(A)$, and if a is fresh for e then $\pi\,(a) \# \pi \bowtie e$.* ■

(Note the equivalence in the first point, which is why we keep a residual empty permutation when we apply a permutation to an expression of the shape $\langle\pi\rangle\, e$.)

5.2.3 Proofs

A generic framework for proofs We now describe a generic framework for defining equational theories over $\mathbb{E}$. Given a relation $Ax \subseteq \mathbb{E} \times \mathbb{E}$, we define the judgement $Ax \vdash e = f$ to hold if it can be inferred in the system displayed in Table 5.1 (where $Ax \vdash e \leq f$ is a shorthand for $Ax \vdash e + f = f$). Notice that we have "hardwired" some laws of Kleene Algebra in this system, on the basis that they should hold for any reasonable equational system for Nominal Kleene Algebra. However, as we have two sets of constants, we should not put inside the generic system the Kleene Algebra laws dealing with them. For instance when we consider expressions over the untyped syntax, the fact that $e \cdot 1 = e$ will be stated inside Ax. It is a simple matter to check that whatever Ax, the relation $Ax \vdash \_ = \_$ is an equivalence relation and $Ax \vdash \_ \leq \_$ is a preorder.

Sets of axioms In Tables 5.2-5.6, we present a number of possible sets of axioms, which may be combined to axiomatise the different subsets we consider. All the axioms displayed here are implicitly universally quantified.

The first groups of axioms correspond to the axioms of KA for 1, declined in a typed and an untyped fashion. We then do the same for 0 and $\bot_A$, first with the KA axioms, and then for their interactions with $\langle\pi\rangle$, ν_a and w_a, always separating between the typed and untyped cases. These sets of axioms for constants are presented in Tables 5.2 and 5.3.

$$\frac{Ax \vdash f = e}{Ax \vdash e = f} \qquad \frac{Ax \vdash e = f \quad Ax \vdash f = g}{Ax \vdash e = f} \qquad \frac{}{Ax \vdash e = f}\ \langle e, f\rangle \in Ax$$

(a) Equivalence and axiom rules.

$$\frac{}{Ax \vdash 0 = 0} \qquad \frac{}{Ax \vdash 1 = 1} \qquad \frac{}{Ax \vdash id_A = id_A} \qquad \frac{}{Ax \vdash \bot_A = \bot_A}$$

$$\frac{}{Ax \vdash l = l} \qquad \frac{}{Ax \vdash a = a} \qquad \frac{Ax \vdash e = g \quad Ax \vdash f = h}{Ax \vdash e + f = g + h} \qquad \frac{Ax \vdash e = g \quad Ax \vdash f = h}{Ax \vdash e \cdot f = g \cdot h}$$

$$\frac{Ax \vdash e = f}{Ax \vdash e^\star = f^\star} \qquad \frac{Ax \vdash e = f}{Ax \vdash \nu_a.e = \nu_a.f} \qquad \frac{Ax \vdash e = f}{Ax \vdash w_a.e = w_a.f} \qquad \frac{Ax \vdash e = f}{Ax \vdash \langle\pi\rangle\, e = \langle\pi\rangle\, f}$$

(b) Congruence rules.

$$\frac{}{Ax \vdash e + f = f + e} \qquad \frac{}{Ax \vdash e + (f + g) = (e + f) + g} \qquad \frac{}{Ax \vdash e + e = e}$$

$$\frac{}{Ax \vdash e \cdot (f + g) = (e \cdot f) + (e \cdot g)} \qquad \frac{}{Ax \vdash (e + f) \cdot e = (e \cdot g) + (f \cdot g)}$$

$$\frac{}{Ax \vdash e \cdot (f \cdot g) = (e \cdot f) \cdot g} \qquad \frac{Ax \vdash f + e \cdot g \leq g}{Ax \vdash e^\star \cdot f \leq g} \qquad \frac{Ax \vdash f + g \cdot e \leq g}{Ax \vdash f \cdot e^\star \leq g}$$

(c) Constant-free Kleene algebra axioms.

Table 5.1: Modular deduction system

We then introduce sets of axioms to handle permutations. The axiom propagating w_a is set apart, as it only makes sense for typed expressions. This group is displayed in Table 5.4. Notice that no law speaks about zeros, as it already has been dealt with in (5.3a).

The sets of axioms in Table 5.5 are simple distributive laws of the restriction and weakening operators. The next group, displayed in Table 5.6, constitutes the core of the nominal theory of expressions. The untyped axioms are mostly the classic nominal axioms, taken from [40]. The only new axiom here is (5.6b), where we use syntactic permutations rather than semantic ones. The typed axioms are for the most part straightforward reformulations of the previous ones. Notice that in the typed case, we do not need to use freshness conditions, but rather typing statements. The last law of the set (5.6f) reflects the fact that for an untyped expression e, if $a \neq b$ then $a \# e \Leftrightarrow a \# \nu_b.e$.

5.2.4 Theories

A *theory* is given by a relation Ax, listing the axioms, and a set S from which we take expressions. As expressions may be typed or untyped, atomic or literate, clean or not and positive or not, there are 16 theories, listed in Table 5.7.

Notice that every subset of expressions mentioned in this table is associated with a single theory. In the following, for concision, we may refer to a theory by simply giving its base set. It is also worth mentioning that for every set S, the theories for $\mathbb{E}_{\mathcal{L}} \cap S$ and $\mathbb{E}_{\mathcal{A}} \cap S$ use the same set of axioms.

The theory $\mathbb{E}_{\mathcal{A}} \cap \mathbb{U} \cap \mathbb{C}$ corresponds precisely to the axiomatisation of NKA given in [40]. In our view, the best theory for defining the interpretation of a program would be $\mathbb{E}^+_{\mathcal{L}} \cap \mathbb{U} \cap \mathbb{C}$, but a relational interpretation is best defined in $\mathbb{E}^+_{\mathcal{L}} \cap \mathbb{T}$.

$$
\begin{aligned}
e \cdot 1 &= e \\
1 \cdot e &= e \\
1 + e \cdot e^\star &\leq e^\star \\
1 + e^\star \cdot e &\leq e^\star
\end{aligned}
$$

(a) Untyped identity axioms

$$
\begin{aligned}
e \cdot id_A &= e && (\text{if } e : A) \\
id_A \cdot e &= e && (\text{if } e : A) \\
id_A + e \cdot e^\star &\leq e^\star && (\text{if } e : A) \\
id_A + e^\star \cdot e &\leq e^\star && (\text{if } e : A)
\end{aligned}
$$

(b) Typed identity axioms

Table 5.2: Identity axioms

$$
\begin{aligned}
e + 0 &= e \\
e \cdot 0 &= e && (\text{if } e \in \mathbb{U}) \\
0 \cdot e &= e && (\text{if } e \in \mathbb{U}) \\
\langle \pi \rangle\, 0 &= 0 \\
\nu_a.0 &= 0
\end{aligned}
$$

(a) Untyped zero axioms

$$
\begin{aligned}
e + \bot_A &= e && (\text{if } e : A) \\
e \cdot \bot_A &= e && (\text{if } e : A) \\
\bot_A \cdot e &= e && (\text{if } e : A) \\
\langle \pi \rangle\, \bot_A &= \bot_{\pi(A)} \\
\nu_a.\bot_A &= \bot_{A \setminus \{a\}} && (\text{if } a \in A) \\
w_a.\bot_A &= \bot_{A \cup \{a\}} && (\text{if } a \notin A)
\end{aligned}
$$

(b) Typed zero axioms

Table 5.3: Zero axioms

$$
\begin{aligned}
\langle \pi \rangle\, (e + f) &= \langle \pi \rangle\, e + \langle \pi \rangle\, f \\
\langle \pi \rangle\, (e \cdot f) &= \langle \pi \rangle\, e \cdot \langle \pi \rangle\, f \\
\langle \pi \rangle\, (e^\star) &= (\langle \pi \rangle\, e)^\star \\
\langle \pi \rangle\, (\nu_a.e) &= \nu_{\pi(a)}.\, \langle \pi \rangle\, e \\
\langle \pi \rangle\, \langle \pi' \rangle\, e &= \langle \pi \circ \pi' \rangle\, e \\
\langle \pi \rangle\, 1 &= 1 \\
\langle \pi \rangle\, id_A &= id_{\pi(A)} \\
\langle () \rangle\, e &= e \\
\langle \pi \rangle\, a &= \pi\,(a) && (\text{if } a \in \mathcal{A}) \\
\langle \pi \rangle\, l &= \pi\,(l) && (\text{if } l \in \mathcal{L})
\end{aligned}
$$

(a) General permutation axioms

$$
\langle \pi \rangle\, (w_a.e) = w_{\pi(a)}.\, \langle \pi \rangle\, e
$$

(b) Permutation vs. w_a

Table 5.4: Permutation axioms

$$
\begin{aligned}
w_a.\,(e + f) &= w_a.e + w_a.f \\
w_a.\,(e^\star) &= (w_a.e^\star) \\
w_a.\,(e \cdot f) &= w_a.e \cdot w_a.f \\
w_a.\,(id_A) &= id_{A \cup \{a\}} && (\text{if } a \notin A) \\
w_a.\,(w_b.e) &= w_b.\,(w_a.e)
\end{aligned}
$$

(a) Weakening

$$
\begin{aligned}
\nu_a.\,(e + f) &= \nu_a.e + \nu_a.f \\
\nu_a.\,(\nu_b.e) &= \nu_b.\,(\nu_a.e)
\end{aligned}
$$

(b) Restriction

Table 5.5: Distributive laws of ν_a, w_a

$$\nu_b.e = \nu_a.(a\ b) \bowtie e \qquad (\text{if } a \# e)$$

(a) Untyped α-conversion with $\bowtie$

$$\nu_b.e = \nu_a.\langle (a\ b)\rangle\, e \qquad (\text{if } a \# e)$$

(b) Untyped α-conversion with $\langle\pi\rangle$

$$\nu_b.e = \nu_a.(a\ b) \bowtie e \quad (\text{if } \nu_b.e : A \text{ and } a \notin A)$$

(c) Typed α-conversion with $\bowtie$

$$\nu_b.e = \nu_a.\langle (a\ b)\rangle\, e \quad (\text{if } \nu_b.e : A \text{ and } a \notin A)$$

(d) Typed α-conversion with $\langle\pi\rangle$

$$\nu_a.e = e \qquad (\text{if } a \# e)$$
$$\nu_a.f \cdot e = \nu_a.(f \cdot e) \quad (\text{if } a \# e)$$
$$e \cdot \nu_a.f = \nu_a.(e \cdot f) \quad (\text{if } a \# e)$$

(e) Untyped nominal axioms

$$\nu_a.w_a.e = e \quad (\text{if } \nu_a.w_a.e : A)$$
$$(\nu_a.f) \cdot e = \nu_a.(f \cdot w_a.e)$$
$$e \cdot (\nu_a.f) = \nu_a.(w_a.e \cdot f)$$
$$\nu_b.w_a.e = w_a.\nu_b.e \quad (\text{if } a \neq b)$$

(f) Typed nominal axioms

Table 5.6: Nominal axioms

NAME	SET	AXIOMS
NKAmpu	$\mathbb{E}^+_{\mathcal{L}} \cap \mathbb{U}$	(5.2a) (5.4a) (5.5b)(5.6b) (5.6e)
NKAspu	$\mathbb{E}^+_{\mathcal{A}} \cap \mathbb{U}$	
NKAnmpu	$\mathbb{E}^+_{\mathcal{L}} \cap \mathbb{U} \cap \mathbb{C}$	(5.2a) (5.5b)(5.6a) (5.6e)
NKAnspu	$\mathbb{E}^+_{\mathcal{A}} \cap \mathbb{U} \cap \mathbb{C}$	
NKAmu	$\mathbb{E}_{\mathcal{L}} \cap \mathbb{U}$	(5.2a)(5.3a)(5.4a) (5.5b)(5.6b) (5.6e)
NKAsu	$\mathbb{E}_{\mathcal{A}} \cap \mathbb{U}$	
NKAnmu	$\mathbb{E}_{\mathcal{L}} \cap \mathbb{U} \cap \mathbb{C}$	(5.2a)(5.3a) (5.5b)(5.6a) (5.6e)
NKAnsu	$\mathbb{E}_{\mathcal{A}} \cap \mathbb{U} \cap \mathbb{C}$	
NKAmpt	$\mathbb{E}^+_{\mathcal{L}} \cap \mathbb{T}$	(5.2b) (5.4a) (5.4b)(5.5a) (5.5b)(5.6d) (5.6f)
NKAspt	$\mathbb{E}^+_{\mathcal{A}} \cap \mathbb{T}$	
NKAnmpt	$\mathbb{E}^+_{\mathcal{L}} \cap \mathbb{T} \cap \mathbb{C}$	(5.2b) (5.5a) (5.5b)(5.6c) (5.6f)
NKAnspt	$\mathbb{E}^+_{\mathcal{A}} \cap \mathbb{T} \cap \mathbb{C}$	
NKAmt	$\mathbb{E}_{\mathcal{L}} \cap \mathbb{T}$	(5.2b)(5.3b)(5.4a) (5.4b)(5.5a) (5.5b)(5.6d) (5.6f)
NKAst	$\mathbb{E}_{\mathcal{A}} \cap \mathbb{T}$	
NKAnmt	$\mathbb{E}_{\mathcal{L}} \cap \mathbb{T} \cap \mathbb{C}$	(5.2b)(5.3b) (5.5a) (5.5b)(5.6c) (5.6f)
NKAnst	$\mathbb{E}_{\mathcal{A}} \cap \mathbb{T} \cap \mathbb{C}$	

Table 5.7: Theories

A difficulty is that if we have a theory $\langle S, Ax\rangle$, with two expressions $e, f \in S$ such that $Ax \vdash e = f$, it may be the case that the proof uses expressions outside of S. This is generally what happens in systems with 0 (or $\bot_A$): if $e \notin S$ and $0 \in S$, then:

$$\frac{\overline{Ax \vdash 0 = 0 \cdot e} \quad \overline{Ax \vdash 0 \cdot e = 0}}{Ax \vdash 0 = 0}.$$

This is a bad property when one wants to prove results by structural induction on proofs. This phenomenon disappears with stable theories:

Definition 5.10 (Stable theory)**.**
A theory $\langle S, Ax\rangle$ is *stable* if for any expressions $e, f \in \mathbb{E}$ such that $Ax \vdash e = f$, $e \in S$ if and only if $f \in S$. ∗

All of our positive theories (those included in $\mathbb{E}^+$) are stable.

5.3 Ordering theories

5.3.1 Definitions

We define two preorders to compare theories. The first one is the strongest one:

Definition 5.11 (Embedding preorder)**.**
A theory $\langle S, Ax\rangle$ *can be embedded into* $\langle S', Ax'\rangle$, written $\langle S, Ax\rangle \preccurlyeq \langle S', Ax'\rangle$ if there is a function φ such that for any $e \in S$, $\varphi(e) \in S'$, and for any $e, f \in S$, $Ax \vdash e = f \Leftrightarrow Ax' \vdash \varphi(e) = \varphi(f)$. In that case we say that φ *is an embedding of* $\langle S, Ax\rangle$ *into* $\langle S', Ax'\rangle$. ∗

When a theory can be embedded into a second one, then every model of the latter one gives rise to a model former one.

However, some intuitively equivalent theory cannot be compared using this preorder. For instance, $\mathbb{E}^+_{\mathcal{A}} \cap \mathbb{T}$ cannot be embedded into $\mathbb{E}^+_{\mathcal{A}} \cap \mathbb{U}$. Indeed, while the typed expressions $id_{\{a\}}$ and $id_{\{a,b\}}$ are not equal (they have different types), they have the same untyped behaviour and both of them should be mapped to the untyped constant 1.

To this end, we introduce a slightly weaker preorder:

Definition 5.12 (Reduction preorder)**.**
A theory $\langle S, Ax\rangle$ *reduces to* $\langle S', Ax'\rangle$, which we denote by $\langle S, Ax\rangle \ll \langle S', Ax'\rangle$, if for any pair $e, f \in S$ there is a pair of expressions $e', f' \in S'$ such that $Ax \vdash e = f \Leftrightarrow Ax' \vdash e' = f'$. ∗

Lemma 5.13. *If* $\langle S, Ax\rangle \ll \langle S', Ax'\rangle$ *and if* $\langle S', Ax'\rangle$ *is decidable, then so is* $\langle S, Ax\rangle$. ■

Remark. This lemma assumes an effective proof of $\langle S, Ax\rangle \ll \langle S', Ax'\rangle$: there must be a way to build the pair e', f' from the pair e, f. Our (Coq) proofs below have this property.

5.3.2 Embeddings

We summarise the results we obtained using Coq on Figure 5.1. (The scripts are available online [14]). A plain arrow is drawn between two theories if the source of the arrow can be embedded into the target of the arrow, and a dashed arrow when the source reduces to the target. Thanks to the decidability result for $\mathbb{E}_{\mathcal{A}} \cap \mathbb{U} \cap \mathbb{C}$ [39], this ensures that all atomic theories are decidable.

We discuss in more details how we obtained some of these results.

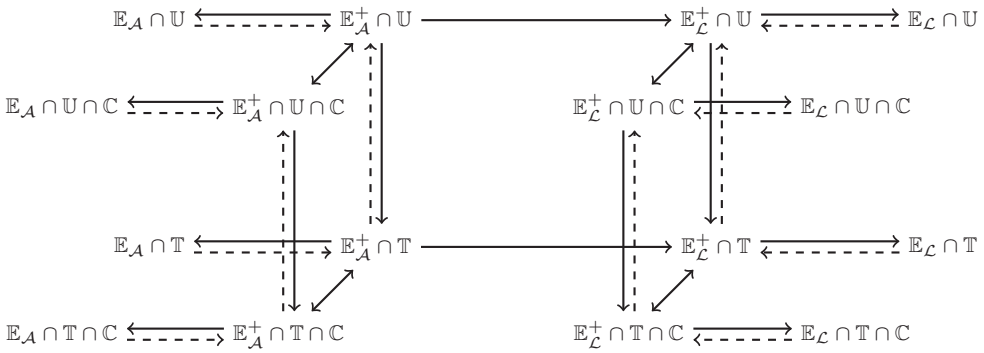


Figure 5.1: The two cubes as sets

Reducing to positive fragments The first step consists in getting rid of the constants 0 and $\bot_A$, so that we can focus on stable theories (Definition 5.10). We only present here the untyped case. In other words we choose a theory $\langle S, Ax\rangle$, with S taken from the set $\{\mathbb{E}_{\mathcal{A}} \cap \mathbb{U}, \mathbb{E}_{\mathcal{A}} \cap \mathbb{U} \cap \mathbb{C}, \mathbb{E}_{\mathcal{L}} \cap \mathbb{U}, \mathbb{E}_{\mathcal{L}} \cap \mathbb{U} \cap \mathbb{C}\}$, the corresponding positive theory being $\langle S \cap \mathbb{E}^+, Ax \setminus (5.3a)\rangle$.

Definition 5.14 (Removing zeros)**.**
If e is an untyped expression, $extract\,(e)$ is the unique normal form of e with respect to the following confluent rewriting system:

$$e + 0 \to e \qquad 0 + f \to f \qquad e \cdot 0 \to 0 \qquad 0 \cdot f \to 0 \qquad \nu_a.0 \to 0 \qquad \langle \pi \rangle\, 0 \to 0 \qquad 0^\star \to 1.$$

*

The interesting property of this function is that if $Ax \vdash e = 0$, then $extract\,(e)$ is syntactically equal to 0, and $extract\,(e) \in \mathbb{E}^+$ otherwise. Furthermore, for every $e \in S$, $Ax \vdash extract\,(e) = e$. The formal proof then relies on two key observations:

1. If $\langle e, f\rangle \in Ax \setminus (5.3a)$, then $Ax \setminus (5.3a) \vdash extract\,(e) = extract\,(f)$.
2. If $\langle e, f\rangle \in (5.3a)$, then $extract\,(e) = extract\,(f)$.

This allows to prove by induction on proofs that:

$$Ax \vdash e = f \Rightarrow Ax \setminus (5.3a) \vdash extract\,(e) = extract\,(f)\,.$$

Because the positive axiomatisation is included in Ax, we also get:

$$Ax \setminus (5.3a) \vdash extract\,(e) = extract\,(f) \Rightarrow Ax \vdash extract\,(e) = extract\,(f)\,.$$

The fact that $extract\,(e)$ is provably equal to e with the axioms Ax allows to close the proof of equivalence, with the entailment:

$$Ax \vdash extract\,(e) = extract\,(f) \Rightarrow Ax \vdash e = f.$$

However, if $Ax \vdash e = 0$ then $extract\,(e) \notin \mathbb{E}^+$. This means that we cannot directly use $extract\,()$ as an embedding between theories. We obtain the reduction as follows: when given the pair e, f, we compute $extract\,(e)$ and $extract\,(f)$. If both of these are equal to zero, then when map the pair to equal positive expressions, say $1, 1$. If both of them are non-zero, then we produce $extract\,(e)\,, extract\,(f)$. Otherwise we produce different positive expressions, say $1, a$ in the atomic case and $1, l$ in the literate case.

From presheaves to freshness, and back Let $\langle S, Ax\rangle$ be a positive untyped theory, meaning $S \subseteq \mathbb{E}^+ \cap \mathbb{U}$, and $\langle S', Ax'\rangle$ be the corresponding positive typed theory. We show here how to transport S into S', and vice versa. This corresponds to the vertical arrows on Figure 5.1. The key tools in this case are the erasure and retyping functions.

Definition 5.15 (Erasure)**.**
The *erasure* of $e \in \mathbb{T}$, written $\lfloor e \rfloor$, is the expression obtained from e by removing all weakenings (w_a), and replacing all id_A with 1 and all $\bot_A$ with 0. ∗

Lemma 5.16. *If $e \in S'$ then $\lfloor e \rfloor \in S$, and if $e : A$ then $\natural(\lfloor e \rfloor) \subseteq A$.* ■

Definition 5.17 (Retyping)**.**
Let $e \in \mathbb{U}$, we define the *retyping of* e, written $\lceil e \rceil$, by structural induction:

$$\lceil 0 \rceil := \bot_\emptyset \qquad \lceil 1 \rceil := id_\emptyset \qquad \lceil a \rceil := a \qquad \lceil l \rceil := l \qquad \lceil e^\star \rceil := \lceil e \rceil^\star$$

$$\lceil e + f \rceil := w_{\natural(f)\setminus\natural(e)}.\lceil e \rceil + w_{\natural(e)\setminus\natural(f)}.\lceil f \rceil \qquad \lceil e \cdot f \rceil := w_{\natural(f)\setminus\natural(e)}.\lceil e \rceil \cdot w_{\natural(e)\setminus\natural(f)}.\lceil f \rceil$$

$$\lceil \langle \pi \rangle\, e \rceil := \langle \pi \rangle \lceil e \rceil \qquad \lceil \nu_a.e \rceil := \nu_a.\lceil e \rceil \ \text{(if } a \in \natural(e)) \qquad \lceil \nu_a.e \rceil := \nu_a.w_a.\lceil e \rceil \ \text{(if } a \notin \natural(e))$$

The notation $w_A.e$ is justified by the law $w_a.w_b.e = w_b.w_a.e$, holding in every typed theory. ∗

Lemma 5.18. *$e \in S$ entails $\lceil e \rceil \in S'$. Furthermore, $\lceil e \rceil$ has the type $\natural(e)$.* ■

These functions allow one to go back and forth between S and S':

Lemma 5.19. *If $e \in S$, then $e = \lfloor \lceil e \rceil \rfloor$. If $e : A$, then:*

$$(5.6f)\ (5.5a)\ (5.4b) \vdash e = w_{A\setminus\natural(\lfloor e \rfloor)}.\lceil \lfloor e \rfloor \rceil\,.$$

Furthermore, if $S \subseteq \mathbb{C}$, we may remove the axiom (5.4b). ■

From this lemma, we obtain that the retyping function is an embedding of S into S'. But it also shows a problem for the other direction. For instance the expressions e and $w_a.e$ have different types, and are thus different, but they will be mapped to the same expression. For this reason, we cannot use the erasure function to embed S' into S.

Nevertheless, we can use it to show that S' reduces to S. Given a pair of expressions $e, f \in S'$, if e and f have the same type, then we produce the pair $\lfloor e \rfloor, \lfloor f \rfloor$ which is equiprovable. If it is not the case, we purposely produce different expressions, as in the previous section.

From atomic to literate We assume there is an atom $\alpha \in \mathcal{A}$ and an element $\lambda \in \mathcal{L}$ with $\natural(\lambda) = \{\alpha\}$. We show the transformation from $\mathbb{E}^+_{\mathcal{A}} \cap \mathbb{U}$ to $\mathbb{E}^+_{\mathcal{L}} \cap \mathbb{U}$, which corresponds to the central horizontal top arrow on Figure 5.1. Let NKApu be the set of axioms $(5.2a), (5.4a), (5.5b), (5.6b), (5.6e)$ corresponding to the theory of these sets.

Definition 5.20 (From atoms to letters.)**.**
Given an expression $e \in \mathbb{E}^+_{\mathcal{A}} \cap \mathbb{U}$, we obtain the expression $\downarrow e \downarrow \in \mathbb{E}^+_{\mathcal{L}} \cap \mathbb{U}$ by replacing every atomic variable a by $(a\ \alpha) \bowtie \lambda$. We write $\downarrow \mathbb{E}^+_{\mathcal{A}} \cap \mathbb{U} \downarrow$ for the set of expressions $f \in \mathbb{E}^+_{\mathcal{L}} \cap \mathbb{U}$, such that there is an expression $e \in \mathbb{E}^+_{\mathcal{A}} \cap \mathbb{U}$ such that $f = \downarrow e \downarrow$. ∗

For any expression e, $e \in \downarrow \mathbb{E}^+_{\mathcal{A}} \cap \mathbb{U} \downarrow$ if and only if for each literate variable l in e there is an atom β such that $\natural(l) = \{\beta\}$ and $(\alpha\ \beta) \bowtie l = \lambda$. This amounts to having l in the orbit of λ. It is also worth noting that $\downarrow \_ \downarrow$ preserves freshness: $a \# e \Leftrightarrow a \# \downarrow e \downarrow$. As for typed and untyped expressions, we define an inverse operation.

Definition 5.21 (Going back)**.**
The inverse operation is only defined on expression from $\lfloor \mathbb{E}^+_{\mathcal{A}} \cap \mathbb{U} \rfloor$, and who thus have a singleton support. The expression $\lceil e \rceil$ is then obtained by replacing every variable by the single atom in its support. ∗

The function $\lceil \_ \rceil$ is the inverse of $\lfloor \_ \rfloor$, $\lfloor \_ \rfloor$ preserves NKApu-equality, and $\lceil \_ \rceil$ preserves NKApu-equality on the image of $\lfloor \_ \rfloor$.

Lemma 5.22. $\forall e \in \mathbb{E}^+_{\mathcal{A}} \cap \mathbb{U}$, NKApu $\vdash e = \lceil \lfloor e \rfloor \rceil$. ■

Lemma 5.23. $\forall e, f \in \mathbb{E}^+_{\mathcal{A}} \cap \mathbb{U}$, *if* NKApu $\vdash e = f$ *then* NKApu $\vdash \lfloor e \rfloor = \lfloor f \rfloor$. ■

Lemma 5.24. $\forall e, f \in \lfloor \mathbb{E}^+_{\mathcal{A}} \cap \mathbb{U} \rfloor$, *if* NKApu $\vdash e = f$ *then* NKApu $\vdash \lceil e \rceil = \lceil f \rceil$. ■

By putting all together, we obtain that $\lfloor \_ \rfloor$ is an embedding of $\mathbb{E}^+_{\mathcal{A}} \cap \mathbb{U}$ into $\mathbb{E}^+_{\mathcal{L}} \cap \mathbb{U}$.

5.4 Relational interpretation of literate expressions

Our main motivation for developing the typed syntax was to define a relational interpretation of expressions. As explained in the introduction, the classical way of interpreting a program as a relation is to consider memory states as functions, associating values to memory cells. A program is then simply a relation between memory states. Furthermore, in most high level programming languages, one cannot access every part of the memory: a variable should be declared before it is used. There are also constructs allowing one to declare a local variable, which is hidden from the rest of the program. Both of these considerations can be encoded by considering functions with a finite domain: the set of memory locations that are visible in the current scope.

Let us be more precise. Consider that the set $\mathcal{A}$ of atoms corresponds to memory locations, and that locations may contain values from an arbitrary set $\mathcal{V}$. A memory state of domain $A \in \mathcal{P}_f(\mathcal{A})$ is then a function from A to $\mathcal{V}$, and an expression of type A will be interpreted as a binary relation over memory states of domain A (whence the presheaf structure).

Regular operations are interpreted using the standard operations on binary relations; in particular, id_A is interpreted as the identity relation on $\mathcal{V}^A$. To interpret letters, we need to fix an equivariant map φ[1] that assign to a letter x a relation between memory states with domain $\natural(x)$. Several choices are possible for the operations of restriction (ν_a) and weakening (w_a), yielding slightly different theories. Here is a possibility which gives rise to a model of our theory: if R is a relation over $\mathcal{V}^A$, then we define

$$\nu_a.R := \left\{ \left\langle f\restriction_{A\setminus\{a\}}, g\restriction_{A\setminus\{a\}} \right\rangle \;\middle|\; \langle f, g\rangle \in R \right\} \;; \qquad \text{(if } a \in A\text{)}$$

$$w_a.R := \{\langle f, g\rangle \mid \langle f\restriction_A, f\restriction_A\rangle \in R \text{ and } f(a) = g(a)\} \;. \qquad \text{(if } a \notin A\text{)}$$

Note that this model is not free: for all relations R, S we have $\nu_a.(R \cdot S) \subseteq (\nu_a.R) \cdot (\nu_a.S)$, which is not an inequation provable from the axioms.

Example. Consider the program $\mathtt{swap}(x, y)$ that exchanges the contents of the variables x and y. The natural implementation of this program is the following:

$$\mathtt{let}\ t\ \mathtt{in}\ t \leftarrow x; x \leftarrow y; y \leftarrow t.$$

The instruction $\mathtt{x} \leftarrow \mathtt{y}$ may be represented by a nominal element $\mathtt{assign}(\mathtt{x}, \mathtt{y})$ with support $\{\mathtt{x}, \mathtt{y}\}$, and such that $\pi(\mathtt{assign}(\mathtt{x}, \mathtt{y})) = \mathtt{assign}(\pi(\mathtt{x}), \pi(\mathtt{y}))$. Accordingly, the program

[1] φ is equivariant if for every $x \in \mathcal{L}$ and every permutation π, $\varphi(\pi(x)) = \pi(\varphi(x))$.

`swap` is represented by the following expression, where the location is hidden using a top-level restriction.

$$\nu_{\mathtt{t}}.\,(\mathtt{assign}\,(\mathtt{t},\mathtt{x})\cdot\mathtt{assign}\,(\mathtt{x},\mathtt{y})\cdot\mathtt{assign}\,(\mathtt{y},\mathtt{t}))\ .$$

Alternatively, one can obtain an expression with a single letter using explicit permutations: let a_1 and a_2 be two atoms, and set $\mathtt{a} := \mathtt{assign}\,(a_1, a_2)$. The instruction $\mathtt{x} \leftarrow \mathtt{y}$ may be represented by $\langle(\mathtt{x}\ a_1)\,(\mathtt{y}\ a_2)\rangle\,\mathtt{a}$, and the program `swap` by

$$\nu_t.\,(\langle(\mathtt{t}\ a_1)\,(\mathtt{x}\ a_2)\rangle\,\mathtt{a})\cdot(\langle(\mathtt{x}\ a_1)\,(\mathtt{y}\ a_2)\rangle\,\mathtt{a})\cdot(\langle(\mathtt{y}\ a_1)\,(\mathtt{t}\ a_2)\rangle\,\mathtt{a})\,.$$

5.5 Future work

We leave two questions for future work. First, is it possible to reduce the literate theory of nominal Kleene algebra to that of atomic nominal Kleene algebra? If not, is there a free language theoretic model for which we could obtain decidability?

Second, is there a free relational model for our literate theory?

List of definitions

1.1 – (Commutative, Idempotent) Monoid 2
1.2 – Group 2
1.3 – (Idempotent) Semi-ring 2
1.4 – Regular expression 3
1.5 – Regular languages 4
1.6 – Finite state automaton 4
1.7 – Relation induced by a letter, by a word 4
1.8 – Path 4
1.9 – Language of an automaton, Recognisable language 5
1.11 – Accessible state, accessible automaton 5
1.16 – Simulation 8
1.17 – Progress, Bisimulation, Bisimilarity 8
1.19 – Recognition by monoid 10
1.20 – Transition monoid 10
1.23 – Syntactic congruence, syntactic monoid 10
1.30 – Kleene Algebra, KA 14

2.6 – Reduction relation 20
2.7 – Closure by $\triangleright$ 20
2.9 – Language $\Gamma(w)$ 20
2.16 – History of a word 25
2.27 – Stratified relation 37
2.28 – Progression, Bisimulation 37

3.2 – Regular graph expressions 44
3.3 – Graph language of an expression, regular sets 44
3.5 – Petri Automaton 45
3.7 – Trace of a run 47
3.8 – Language of a Petri automaton, recognisable sets 47
3.9 – Proper run 47
3.15 – $\mathscr{A}(e)$ 51
3.18 – Box 52
3.19 – Identity box 53
3.20 – Composition of boxes 53
3.22 – Type 55
3.23 – Typed boxes 55
3.27 – (Finite) Template 57
3.28 – Boxes generated by a template 57
3.29 – Regular operations on templates 57
3.30 – support 58
3.31 – Atomic box, atomic template 58
3.36 – Box of a transition 60
3.40 – $\mathscr{A}$-validity 62
3.42 – Branching automaton 63
3.43 – Runs and language of a branching automaton 63
3.44 – Term language of an expression 64

4.1 – Allegoric regular expressions 68
4.2 – Graph of a ground term: $\mathcal{G}(w)$ 68
4.3 – Graph homomorphism, preorders $\blacktriangleleft$ and $\triangleleft$ 68
4.6 – Terms of an expression 69
4.7 – Graphs of an expression 69
4.10 – Converse-normal form 71
4.11 – Translation between allegoric and graph expressions 71
4.12 – $\lceil G \rceil$ 71
4.16 – Reading, language of a run 72
4.18 – Language recognised by a Petri automaton 73
4.24 – Simulation 79
4.25 – Embedding 79
4.28 – Exchangeable transitions 80

5.1 – Nominal expressions 88
5.2 – Untyped expression 88
5.3 – Freshness 89
5.4 – Typed expressions 89
5.5 – Atomic expressions, literate expressions 89
5.6 – Positive expression 89
5.7 – Action of a permutation on an expression 90
5.8 – Clean expressions 90
5.10 – Stable theory 94
5.11 – Embedding preorder 94
5.12 – Reduction preorder 94
5.14 – Removing zeros 95
5.15 – Erasure 96
5.17 – Retyping 96
5.20 – From atoms to letters. 96
5.21 – Going back 97

Author's Contributions

[1] Paul Brunet and Damien Pous. "Kleene Algebra with Converse". In: *Proceedings RAMiCS*. Ed. by Peter Höfner, Peter Jipsen, Wolfram Kahl, and Martin Eric Müller. Vol. 8428. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 101–118. ISBN: 978-3-319-06250-1. DOI: 10.1007/978-3-319-06251-8_7.

[2] Paul Brunet and Damien Pous. "Decidability of Identity-free Relational Kleene Lattices". In: *Proceedings JFLA*. Ed. by David Baelde and Jade Alglave. Le Val d'Ajol, France, Jan. 2015.

[3] Paul Brunet and Damien Pous. "Petri Automata for Kleene Allegories". In: *Proceedings LICS*. July 2015, pp. 68–79. DOI: 10.1109/LICS.2015.17.

[4] Paul Brunet and Damien Pous. "A formal exploration of Nominal Kleene Algebra". In: *Proceedings MFCS*. Ed. by Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Aug. 2016.

[5] Paul Brunet and Damien Pous. "Algorithms for Kleene algebra with converse". In: *Journal of Logical and Algebraic Methods in Programming* 85.4 (2016). Relational and algebraic methods in computer science, pp. 574–594. ISSN: 2352-2208. DOI: http://dx.doi.org/10.1016/j.jlamp.2015.07.005.

[6] Paul Brunet, Damien Pous, and Insa Stucke. "Cardinalities of Relations in Coq". In: *Proceedings ITP*. Ed. by Jasmin Christian Blanchette and Stephan Merz. Lecture Notes in Computer Science. Springer, Aug. 2016.

Bibliography

[1] Parosh Aziz Abdulla, Yu-Fang Chen, Lukáš Holík, Richard Mayr, and Tomáš Vojnar. "When Simulation Meets Antichains". In: *Proceedings TACAS*. Vol. 6015. Lecture Notes in Computer Science. Springer Verlag, 2010, pp. 158–174 (cit. on p. 36).

[2] Hajnal Andréka and Dmitry A. Bredikhin. "The equational theory of union-free algebras of relations". In: *Algebra Universalis* 33.4 (1995), pp. 516–532. ISSN: 0002-5240 (cit. on pp. vi, 41, 68–69).

[3] Hajnal Andréka, Szabolcs Mikulás, and István Németi. "The equational theory of Kleene lattices". In: *Theoretical Computer Science* 412.52 (2011), pp. 7099–7108 (cit. on pp. vi, 68–70).

[4] Valentin M. Antimirov. "Partial Derivatives of Regular Expressions and Finite Automaton Constructions". In: *Theoretical Computer Science* 155.2 (1996), pp. 291–319 (cit. on pp. 12, 37).

[5] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1999. ISBN: 9780521779203 (cit. on p. 23).

[6] Stephen L. Bloom, Zoltán Ésik, and Gheorghe Stefanescu. "Notes on equational theories of relations". In: *Algebra Universalis* 33.1 (1995), pp. 98–126 (cit. on pp. vi, 17–25, 29, 33–35, 39, 67, 71).

[7] Maurice Boffa. "Une remarque sur les systèmes complets d'identités rationnelles". In: *Informatique Théorique et Applications* 24 (1990), pp. 419–428 (cit. on p. 14).

[8] Filippo Bonchi, Marcello M. Bonsangue, Jan J. M. M. Rutten, and Alexandra Silva. "Logic and Program Semantics: Essays Dedicated to Dexter Kozen on the Occasion of His 60th Birthday". In: ed. by Robert L. Constable and Alexandra Silva. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. Chap. Brzozowski's Algorithm (Co)Algebraically, pp. 12–23. ISBN: 978-3-642-29485-3. DOI: 10.1007/978-3-642-29485-3_2 (cit. on p. 7).

[9] Filippo Bonchi and Damien Pous. "Checking NFA equivalence with bisimulations up to congruence". In: *Proceedings POPL*. ACM, 2013, pp. 457–468. ISBN: 978-1-4503-1832-7 (cit. on pp. 18, 36–39).

[10] Francis Bossut, Max Dauchet, and Bruno Warin. "A Kleene theorem for a class of planar acyclic graphs". In: *Information and Computation* 117.2 (1995), pp. 251–265 (cit. on p. 66).

[11] Thomas Braibant and Damien Pous. "Deciding Kleene Algebras in Coq". In: *Logical Methods in Computer Science* 8.1 (2012), pp. 1–16 (cit. on p. 39).

[12] Paul Brunet. *KAC software*. http://perso.ens-lyon.fr/paul.brunet/kac. 2014 (cit. on pp. 23, 39).

[13] Paul Brunet. *RKLM software*. http://perso.ens-lyon.fr/paul.brunet/rklm. 2014 (cit. on p. 82).

[14] Paul Brunet. *Coq library of NKA proofs*. http://perso.ens-lyon.fr/paul.brunet/nka. 2016 (cit. on p. 94).

[15] Paul Brunet and Damien Pous. "Kleene Algebra with Converse". In: *Proceedings RAMiCS*. Ed. by Peter Höfner, Peter Jipsen, Wolfram Kahl, and Martin Eric Müller. Vol. 8428. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 101–118. ISBN: 978-3-319-06250-1. DOI: 10.1007/978-3-319-06251-8_7 (cit. on p. 67).

[16] Paul Brunet and Damien Pous. "Petri Automata for Kleene Allegories". In: *Proceedings LICS*. July 2015, pp. 68–79. DOI: 10.1109/LICS.2015.17 (cit. on pp. vi, 41).

[17] Paul Brunet and Damien Pous. "A formal exploration of Nominal Kleene Algebra". In: *Proceedings MFCS*. Ed. by Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Aug. 2016 (cit. on p. vi).

[18] Paul Brunet and Damien Pous. "Algorithms for Kleene algebra with converse". In: *Journal of Logical and Algebraic Methods in Programming* 85.4 (2016). Relational and algebraic methods in computer science, pp. 574–594. ISSN: 2352-2208. DOI: http://dx.doi.org/10.1016/j.jlamp.2015.07.005 (cit. on p. vi).

[19] Janusz A Brzozowski. "Canonical regular expressions and minimal state graphs for definite events". In: *Mathematical theory of Automata* 12.6 (1962), pp. 529–561 (cit. on p. 7).

[20] Janusz A. Brzozowski. "Derivatives of regular expressions". In: *Journal of the ACM* 11 (1964), pp. 481–494 (cit. on p. 12).

[21] Olivier Carton. *Langages formels, calculabilité et complexité*. Vol. 101. Vuibert, 2008 (cit. on p. 3).

[22] John Horton Conway. *Regular algebra and finite machines*. Chapman and Hall Mathematics Series, 1971 (cit. on pp. 3, 13, 67).

[23] Bruno Courcelle and Joost Engelfriet. *Graph structure and monadic second-order logic. A language-theoretic approach.* Encyclopedia of Mathematics and its applications, Vol. 138. Collection Encyclopedia of Mathematics and Applications, Vol. 138. Cambridge University Press, June 2012, p. 728 (cit. on p. 66).

[24] Laurent Doyen and Jean-François Raskin. "Antichain Algorithms for Finite Automata". In: *Proceedings TACAS*. Vol. 6015. Lecture Notes in Computer Science. Springer Verlag, 2010, pp. 2–22 (cit. on p. 36).

[25] Zoltán Ésik. "Group Axioms for Iteration". In: *Information and Computation* 148.2 (1999), pp. 131–180. ISSN: 0890-5401. DOI: 10.1006/inco.1998.2746 (cit. on p. 14).

[26] Zoltán Ésik and Laszlo Bernátsky. "Equational properties of Kleene algebras of relations with conversion". In: *Theoretical Computer Science* 137.2 (1995), pp. 237–251 (cit. on pp. vi, 17–18, 20, 35, 39, 67).

[27] Peter J. Freyd and Andre Scedrov. *Categories, Allegories*. North Holland, 1990 (cit. on pp. vi, 41, 67–69).

[28] Murdoch James Gabbay and Vincenzo Ciancia. "Freshness and Name-Restriction in Sets of Traces with Names". In: *Proceedings FoSSaCS*. Springer, 2011, pp. 365–380 (cit. on pp. vi, 87).

[29] Murdoch Gabbay and Andrew M Pitts. "A new approach to abstract syntax involving binders". In: *Proceedings LICS*. IEEE. 1999, pp. 214–224 (cit. on p. 88).

[30] Victor M. Glushkov. "The abstract theory of automata". In: *Russian Mathematical Surveys* 16.5 (1961), p. 1 (cit. on pp. 12, 35).

[31] Ursula Goltz and Wolfgang Reisig. "The non-sequential behaviour of Petri nets". In: *Information and Control* 57.2 (1983), pp. 125–147 (cit. on p. 86).

[32] John E. Hopcroft and Richard M. Karp. *A linear algorithm for testing equivalence of finite automata*. Tech. rep. Cornell University, 1971 (cit. on pp. 36, 38–39).

[33] Lucian Ilie and Sheng Yu. "Follow automata". In: *Information and Computation* 186.1 (2003), pp. 140–162. ISSN: 0890-5401. DOI: http://dx.doi.org/10.1016/S0890-5401(03)00090-7 (cit. on p. 12).

[34] Lalita Jategaonkar and Albert R. Meyer. "Deciding true concurrency equivalences on safe, finite nets". In: *Theoretical Computer Science* 154.1 (1996). Twentieth International Colloquium on Automata, Languages and Programming, pp. 107–143. ISSN: 0304-3975. DOI: http://dx.doi.org/10.1016/0304-3975(95)00132-8 (cit. on pp. 83, 85–86).

[35] Stephen Cole Kleene. *Representation of Events in Nerve Nets and Finite Automata*. Memorandum. Rand Corporation, 1951 (cit. on pp. v, 13, 67).

[36] Dexter Kozen. "A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events". In: *Proceedings LICS*. IEEE Computer Society, 1991, pp. 214–225 (cit. on pp. 14–15, 20, 67).

[37] Dexter Kozen. "Kleene algebra with tests". In: *Transactions on Programming Languages and Systems* 19.3 (May 1997), pp. 427–443. DOI: 10.1145/256167.256195 (cit. on p. 87).

[38] Dexter Kozen. *Typed Kleene algebra*. Tech. rep. Cornell University, 1998 (cit. on p. 56).

[39] Dexter Kozen, Konstantinos Mamouras, Daniela Petrisan, and Alexandra Silva. "Nominal Kleene Coalgebra". In: *Proceedings ICALP*. Springer, 2015, pp. 286–298 (cit. on pp. vi, 87, 94).

[40] Dexter Kozen, Konstantinos Mamouras, and Alexandra Silva. "Completeness and Incompleteness in Nominal Kleene Algebra". In: *Proceedings RAMiCS*. Springer, 2015, pp. 51–66 (cit. on pp. vi, 87, 91).

[41] Daniel Krob. "A Complete System of B-Rational Identities". In: *Proceedings ICALP*. Vol. 443. Lecture Notes in Computer Science. Springer Verlag, 1990, pp. 60–73 (cit. on pp. 14, 20, 67).

[42] Kamal Lodaya and Pascal Weil. "Series-parallel posets: Algebra, automata and languages". In: *Proceedings STACS*. Ed. by Michel Morvan, Christoph Meinel, and Daniel Krob. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 555–565. ISBN: 978-3-540-69705-3. DOI: 10.1007/BFb0028590 (cit. on p. 62).

[43] Kamal Lodaya and Pascal Weil. "Series–parallel languages and the bounded-width property". In: *Theoretical Computer Science* 237.1 (2000), pp. 347–380. ISSN: 0304-3975. DOI: http://dx.doi.org/10.1016/S0304-3975(00)00031-1 (cit. on p. 62).

[44] Kamal Lodaya and Pascal Weil. "Rationality in Algebras with a Series Operation". In: *Information and Computation* 171.2 (2001), pp. 269–293. ISSN: 0890-5401. DOI: http://dx.doi.org/10.1006/inco.2001.3077 (cit. on p. 62).

[45] Robert McNaughton and Hideki Yamada. "Regular Expressions and State Graphs for Automata". In: *IRE Transactions on Electronic Computers* EC-9.1 (Mar. 1960), pp. 39–47. ISSN: 0367-9950. DOI: 10.1109/TEC.1960.5221603 (cit. on p. 12).

[46] Albert R. Meyer and Larry J. Stockmeyer. "The equivalence problem for regular expressions with squaring requires exponential space". In: *Proceedings SWAT*. IEEE. 1972, pp. 125–129 (cit. on pp. 13, 35, 67, 83).

[47] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989 (cit. on p. 8).

[48] Bernhard Möller. "Typed Kleene Algebras". In: *Proceedings MPC*. Lecture Notes in Computer Science. Citeseer. 1999 (cit. on p. 56).

[49] Tadao Murata. "Petri nets: Properties, analysis and applications". In: *Proceedings of the IEEE* 77.4 (Apr. 1989), pp. 541–580. ISSN: 0018-9219. DOI: 10.1109/5.24143 (cit. on p. 85).

[50] Jean-Éric Pin. *Mathematical Foundations of Automata Theory*. Lecture notes. Oct. 2015 (cit. on p. 10).

[51] Andrew M Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Vol. 57. Cambridge University Press, 2013 (cit. on p. 88).

[52] Damien Pous. "Kleene Algebra with Tests and Coq Tools for While Programs". In: *Proceedings ITP*. Vol. 7998. Lecture Notes in Computer Science. Springer Verlag, 2013, pp. 180–196 (cit. on p. 39).

[53] Damien Pous and Davide Sangiorgi. "Advanced Topics in Bisimulation and Coinduction". In: Cambridge University Press, 2011. Chap. about "Enhancements of the coinductive proof method", pp. 233–289. ISBN: 9781107004979 (cit. on pp. 9, 18, 37).

[54] Vaughan Pratt. "Dynamic algebras: Examples, constructions, applications". In: *Studia Logica* 50.3 (1991), pp. 571–605. ISSN: 1572-8730. DOI: 10.1007/BF00370685 (cit. on p. 15).

[55] Volodimir Nikiforovych Redko. "On defining relations for the algebra of regular events". In: *Ukrainskii Matematicheskii Zhurnal* (1964), pp. 120–126 (cit. on pp. 13, 67).

[56] Jacques Sakarovitch. *Elements of automata theory*. Cambridge University Press, 2009 (cit. on p. 3).

[57] Arto Salomaa. "Two Complete Axiom Systems for the Algebra of Regular Events". In: *Journal of the ACM* 13.1 (1966), pp. 158–169 (cit. on p. 13).

[58] Davide Sangiorgi. "On the Bisimulation Proof Method". In: *Mathematical Structures in Computer Science* 8 (1998), pp. 447–479 (cit. on pp. 9, 37).

[59] Walter J Savitch. "Relationships between nondeterministic and deterministic tape complexities". In: *Journal of computer and system sciences* 4.2 (1970), pp. 177–192 (cit. on p. 83).

[60] Ken Thompson. "Regular Expression Search Algorithm". In: *Communications of the ACM* 11 (1968), pp. 419–422 (cit. on pp. 12, 51, 84).

[61] Jacobo Valdes, Robert E. Tarjan, and Eugene L. Lawler. "The Recognition of Series Parallel Digraphs". In: *Proceedings STOC*. STOC '79. Atlanta, Georgia, USA: ACM, 1979, pp. 1–12. DOI: 10.1145/800135.804393 (cit. on pp. 42, 54).

[62] Martin De Wulf, Laurent Doyen, Thomas A. Henzinger, and Jean-Francois Raskin. "Antichains: A New Algorithm for Checking Universality of Finite Automata". In: *Proceedings CAV*. Vol. 4144. Lecture Notes in Computer Science. Springer Verlag, 2006, pp. 17–30 (cit. on p. 36).