Representation and Analyses of Web Content and Processing

Nabil Layaïda

HABILITATION À DIRIGER DES RECHERCHES

présentée par

Nabil LAYAÏDA

pour obtenir le diplôme
d'Habilitation à Diriger Des Recherches
de l'Université de Grenoble

(Spécialité: Informatique)

Representation and Analyses of Web Content and Processing

Date de soutenance : 23 Avril 2013

Composition du jury :	Président	DR. Jérôme Euzenat
	Rapporteurs	Prof. Rachid Guerraoui
		Prof. Sophie Tison
		DR. Giuseppe Castagna
	Examinateurs	Prof. Hassan Aït-Kaci
		Prof. Mohand-Saïd Hacid
		Prof. Véronique Benzaken
		DR. Vincent Quint

Travaux de recherche effectués au sein de l'équipe-projet WAM conjointe à Inria Grenoble – Rhône-Alpes et au Laboratoire d'Informatique de Grenoble (LIG).

Contents

One

Introduction

1.1 Objectives

The goal of my research work is the design and implementation of a new generation of web applications where content is made richer and long lasting and processing made simpler, more flexible, safer and more effective. Designing models consists of defining document domain specific languages capable of describing and representing a wide variety of electronic documents together with their processing. By variety, I mean document classes ranging from traditional documents such as books, technical manuals, scientific articles, web applications, multimedia presentations up to active and interactive documents combining video, audio, 3D objects, RSS feeds, blogs, forms. Such documents can be active and reactive to user and program inputs and may potentially live in a complex distributed computing environment made of remote application servers and databases. Today, this variety keeps increasing as the result of innovations made both by new generations of web applications and also by the multiplication of distribution channels such as mobile devices, set top boxes, interactive television, tablets, etc.

One of the main requirements for document representation is independence from the manipulating software and the underlying hardware architecture, for obvious reasons related to their sustainability. It must also satisfy long-term access constraints and storage and need to have high readability to facilitate their use and repurposing in future applications. These constraints are so crucial that document representation is implicitly expected to be rather descriptive, rich enough to cover a wider range of document classes while being sufficiently flexible to accommodate future uses that are unknown at design time.

Similar requirements are expected for document processing. Document manipulation programs are characterized by a set of "typical" tasks, which consist in updating, transmitting, displaying, transforming, adapting, repurposing, or publishing documents on a given infrastructure. One of the peculiarities of these tasks is the predominance of tree and graphical structure manipulation, either to update the content or its displayed form. Languages such as XSLT or XQuery have been designed to serve as domain specific languages for such manipulations. They have been both equipped with tree querying mechanisms which allow selecting, extracting and combining tree fragments using regular path expressions, via XPath. Additionally, content models are often described

using regular tree constraints (XML Schemas) [Bray et al., 2004] that each document instance either read or produced must adhere to. Regularity was introduced in content representation and processing to provide more abstract representations and powerful processing. However, a lack of appropriate supporting tools often resulted in complex programs that are difficult to maintain and make evolve, vulnerable to errors and resource hungry, both in terms of memory and speed.

Today, web content and applications are becoming the main interface to performing all kinds of daily tasks such as stating and paying government taxes, booking plane or train tickets, planning holidays, shopping, managing bank accounts, etc. As a consequence, it is becoming increasingly important to study their foundations, enhance their capabilities, facilitate their design, verify their correctness, optimize them automatically and make them more flexible and adaptive. The present work aims at contributing to these objectives and to a new generation of content models and programming languages. Since standards are key success factor in this research area, I do actively contribute to web standards either by proposing new ones such as for SMIL [Layaïda & Ossenbruggen, 2001] or by enhancing existing ones such as XML and XQuery[Boag et al., 2006].

1.2 Overview of the research area

The hypermedia research area has experienced considerable mutations over the last twenty years. Just before the web was born, it was easier to identify the scientific community, its main conferences such as ACM Hypertext, Electronic Publishing, ECHT, and the overall scientific topics. Starting from the early 1990s, and the exponential growth and impact brought by the web success, addressed topics became much broader in scope and progressively encompassed other areas of research. Today the World Wide Web Conference scientific topics include human machine interfaces, knowledge representation, machine learning, networking, information retrieval, databases, programming languages, social networks, etc. Even some fundamental subjects related to logical reasoning, foundations of databases and language theory are being revisited for the web development.

To better understand current scientific trends and challenges, one needs to examine closely the initial design goals of the World Wide Web project, which was initiated and developed by Tim Berners-Lee and Robert Cailliau at CERN. Beyond the development of a global hypertext system, simpler design principles compared to those prevailing in the literature at that time were made the priority. More precisely, systems such as Dexter [Halasz & Schwartz, 1994], NoteCards, Intermedia or Hyper-G [Conklin, 1987; Kappe et al., 1993; Akscyn et al., 1988], reference models in the community, were considered by the web inventors to be too centralized and too closed [Berners-Lee et al., 1992]. As a matter of fact, the rule in hypertext was monolithic design based on a single integrated application built on top of a centrally managed and uniform storage system. These systems favored a disciplined use of content by enforcing a

stricter consistency of hyperlinks relating pieces of information, stronger syntactic rules for content description and publication (entities created in theses system are uniquely identified by universal names issued by a central authority). In contrast, the web architecture was designed to be highly decentralized, open and based on a number of shared, application independent principles. Resources and links between them can be created and added freely without any guarantees on resource availability, or on document and link consistency. These principles have resulted in the development of three key components that are the cornerstones of the web:

- **A universal resource naming and addressing scheme** for locating and retrieving hypertext documents, images, emails, indexes, files, etc.: the so-called universal resource identifiers or URIs.
- **A hypertext document format**: HTML, the hypertext markup language. HTML is a text based format with presentation oriented markup or tagging simple enough to be used by novice users.
- **A network communication protocol**: the hypertext transport protocol HTTP which is a stateless TCP/IP based protocol for retrieving remote resources.

The notion of sharing described by Berners-Lee corresponds to the notion of standard as practiced by organizations such as ISO, IETF and later by the W3C created specifically for the web. Standards where usually intended to achieve technically, by means of consensus or vote, the technical conditions of interoperability between systems for mature technology. In contrast, web formats, protocols and languages were designed in a very prospective manner meant to guide future developments and to "realize the full potential of the web". This is particularly exemplified by the Semantic Web activity, database oriented programming languages such as XQuery [Boag et al., 2006] or mobile web initiatives launched well ahead of industry demand.

Interestingly, content, one of the main building blocks of the web did have only a limited success in its development. Putting aside the fierce browser war and patent disputes, some initiatives such XHTML and CDF (Compound documents Formats) faced strong challenges and were later simply discontinued, despite user demand for richer content. XHTML was an attempt to reinforce syntactic correctness by reformulating HTML in XML while the CDF initiative was more ambitious. It targeted the design of a more advanced document model which accounts for the different content facets by promoting the integration of several specialized languages developed independently, namely HTML, SVG, SMIL and XForms. The idea in combining such formats is to answer the growing needs and user demands for richer Web applications and content. The idea was also to ensure that content and the skills required to produce it remain uniform on all platforms, ubiquitous, accessible to all, and cost effective. However, unlike HTML, most of these languages are XML-based and

each one comes with its own schema that can be regarded as functional grammar that pertains to a specific area of functionality: for example, SVG enables vector graphics; XForms addresses form input collection and submission; SMIL describes temporal synchronization. Most existing web applications today, if represented using CDF, would use at least a combination of two or more of these languages. One of the major scientific challenges was to find appropriate means to produce new languages, by composition of separately defined ones. While the difficulty may seem simple since syntactic, there is no clear method today to achieve such a goal easily. At semantic level, it is even more challenging. Defining a uniform presentational model which accounts for such a rich content while supporting advanced and consistent styles and exposing appropriate APIs for content manipulating programs is today beyond reach.

From an architectural point of view, most of existing web applications can easily fit the typical three-tier architecture (user interface, application and storage). However, a closer look at how the different tiers are designed reveals serious problems and a major «impedance mismatch» problem [Lämmel & Meijer, 2007]. First, a web page is no longer guaranteed to display or to behave uniformly on the already fairly large variety of platforms. The complexity and burden to manage such complexity has been progressively transferred to the UI designers, which are required to implement specific functions for each one of them. Second, the application tier where content and data are processed and stored is facing a triple collision of, hard to reconcile, data models: documents structures (trees), programming languages (often object-oriented) and databases (often relational). Most of the issues raised above are today subcontracted to low-level programming such as JavaScripting or to a jungle of frameworks. This comes at the cost of severely compromising accessibility, maintainability, reliability, security, and performance and contributes greatly to web fragmentation.

In summary, the web has badly accommodated an inadequate infrastructure, mainly obtained by ad hoc extensions. This reveals the weakest component in its initial design: content. As a result, appropriate abstractions are still needed to make it richer, more reliable, secure, efficient and flexible. In the recent period, there are clear signs of a renewed interest and increasing activity both in content representation as witnessed by HTML5 and in web programming through XQuery3 [Engovatov & Robie, 2010].

1.3 Document Organization

After this introduction, this document is organized as follows:

Chapter 2, presents a brief introduction to electronic documents, the separation principle together with the main document dimensions. Then I summarize the research topics I addressed and the contributions I made since I was hired as a researcher at Inria. For each of these contributions, I give references to relevant publications describing in more details the results and, when applicable, I indicate the names of collaborators involved in the research work together with those of supervised PhD students.

In **Chapter** 3, I give a curriculum summarizing the theses I supervised, the projects and contracts that funded my research work, administrative responsibilities, community involvement, technology transfer, standardization and software development. My teaching activities as well as the complete list of my scientific publications are given in the last part of this chapter. In the remaining part of the document, I present in detail only a sample of my most recent research work related to XML reasoning. This work is described through a selection of three extended articles focusing on XML static analyses, XML schema evolution management in programs and finally, a more advanced XML logic featuring functions and parametric subtyping.

Chapter 4 introduces a logic for reasoning over finite trees, a sound and complete decision procedure for checking the satisfiability of a formula of the logic as well as its effective implementation. The logic is a variant of the μ-calculus adapted for finite trees and equipped with backward modalities and nominals. Specifically, the logic is an alternation-free modal μ-calculus with converse, where formulas are cycle-free, and which is interpreted over finite ordered trees. The time complexity of the satisfiability-testing algorithm is optimal: $2^{\mathcal{O}(n)}$ in terms of formula size n. I present crucial implementation techniques like the use of symbolic techniques (BDD) and heuristics used to make the algorithm as fast as possible in practice. Our implementation is available online, and can be used to solve logical formulas of practically significant size.

In **Chapter** 5, the problem of XML Schema evolution is studied. In the ever-changing context of the web, XML schemas continuously change in order to cope with the natural evolution of the entities they describe. Schema changes have important consequences. First, existing documents valid with respect to the original schema are no longer guaranteed to fulfill the constraints described by the evolved schema. Second, the evolution also impacts programs manipulating documents whose structure is described by the original schema. I propose a unifying framework for determining the effects of XML Schema evolution both on the validity of documents and on queries. The system is very powerful in analyzing various scenarios in which forward/backward compatibility of schemas is broken, and in which the result of a query may not be anymore what was expected. Specifically, the system offers a predicate language which allows one to formulate properties related to schema evolution. The system then relies on exact reasoning techniques introduced in the previous Chapter. The system was tested with real-world use cases, in particular with the main standard document formats used on the web, as defined by W3C. The system identifies precisely compatibility relations between document formats. In case these relations do not hold, the system can identify queries that must be reformulated in order to produce the expected results across successive schema versions.

Chapter 6 considers a type algebra equipped with recursive, product, function, intersection, union, and complement types together with type variables and implicit universal quantification over them. In particular, I focus on the

subtyping relation recently defined by Castagna and Xu [Castagna & Xu, 2011] over such type expressions and show how this relation can be decided in $2^{\mathcal{O}(n)}$, answering an open question. The novelty, originality and strength of the proposed solution reside in introducing a logical modeling for the semantic subtyping framework. The semantic subtyping is modeled in a tree logic and satisfiability-testing algorithm is used in order to decide subtyping. I report on practical experiments made with a full implementation of the system. This provides a powerful polymorphic type system aiming at maintaining full static type-safety of functional programs that manipulate trees, even with higher-order functions, which is particularly useful in the context of XML.

Chapter 7 concludes this work and draws some perspectives for the future.

Two

Research summary

In order to define the vocabulary used in this document, I first define what I mean by document. The word comes from the Latin *documentum* which has a root from *docere*, which means teaching. A document is an object carrying meaning used to transmit knowledge. A document is often not dissociated from the media on which it is conveyed or exchanged, the medium such as paper or a computer file, giving it a form of continuity or stability over time. The information carried on this medium is usually encoded as a variety of more or less complex symbols such as characters, glyphs, figures, diagrams, tables, annotations, etc. Because of the physical connotation attached to the medium and symbols, the amount of information contained in a document is often considered as finite or at least clearly delimited.

Traditionally, a document is created by an author or a group of authors and consumed by readers. This separation of roles is however less clear today. Collaborative platforms such as wikis allow a group of users to concurrently author and read documents collectively. More generally, the acceleration of the virtualization in the Web era has profoundly modified not only editorial chains but also the precise meaning attributed to the word document. In the sequel, the term can sometimes mean a video enriched with textual annotations and images, it can also mean real-time flow of information news, tweets, blogs, dynamically generated documents from user profiles, etc.

The first generation of electronic document management systems were designed to reproduce the classical document preparation tasks on computer systems such as copy/pasting, formatting and typesetting, correcting, annotating and archiving. The final goal of the entire chain was often publishing on paper, web sites or other media such as portable document format (pdf) files or CD Rom. Today, a large amount of documents are dynamic and some have a very short lifecycle such as tweets. In addition, some documents containing time-based media such animated graphics, audio and video are consumed only on electronic devices. As a result, the concept of document becomes difficult to identify because one can no longer refer to specific characteristics such as the basic elements that compose it, or its stability over time (its persistent nature).

Today, document addresses on the web can refer to, in an indistinguishable manner, stable, slow evolving or dynamically generated content obtained via complex web services invocations such as REST [Fielding, 2000]. Similarly, documents do frequently embed programs that perform all kinds of tasks such

as modifying the content in a context sensitive manner depending on the user location, language and other parameters. The boundary between content and computations is becoming fuzzy and the various attempts to establish standardized interfaces between them (such as DOM) did so far have only little success. Such interactions are often bypassed by low-level script manipulations violating a key principle of electronic documents: the separation principle.

2.1 Separation principle

One of the fundamental principles of electronic documents, which emerged in the late 60s, is the separation principle. This principle consists in describing separately the structure and content of a document (headings, chapters, sections, paragraphs, links, etc.) from its presentation or physical form (on the screen, on paper, and so on). This principle has been introduced to meet several needs:

- **Ensure that documents are long lasting:** a more direct access to content and document organization allows easier maintenance and conversion against possible technology changes.
- **Facilitate publishing on various media** or in different forms or variants of the same document when needed.
- **Facilitate exchange, update and repurposing** of documents between users and applications. Often documents are the result of successive updates made by users and applications from different organizations.

The separation principle is put into practice by syntactic techniques, which allows highlighting the structure of a document: it is the tagging or marking process. Tagging, which appeared for electronic documents in the late 70s, consists of placing marks around textual elements. It comes in three distinct flavors:

1. The presentational mark up which corresponds to codes which produce a visual effect on the display or paper: for example making some text fragment render in bold, italic or in particular font, etc.
2. Procedural mark up, whose primitive elements are presentation functions. These functions are enriched, in a procedural programming environment, by a stack of more complex routines or macros that produce complex presentation constructions: troff, TEX, LATEX are examples of such marking.
3. Descriptive mark up describes what the marked text is meant to represent rather than how to display or process it by means of labels or tags. Labels are text marks carrying a description, whose only reading is supposed to suggest what the marked item represents: chapter, title, section, paragraph, appendix, etc.

Descriptive markup is used as the primary technique to ensure the separation principle. It is indeed possible to obtain formatted documents, i.e. having a publishable form, by enriching a document marked up descriptively via styles attachment or by conversion (transformation) to a presentational format. A typical example of such operation are CSS attachment to XML content or the conversion of documents from the DocBook DTD [Walsh & Muellner, 1999] to XSL-FO [Berglund, 2006] for printing or to HTML for browser display. Descriptive markup was the main motivation behind the introduction of meta-languages such as GML [Goldfarb, 1996], SGML [ISO-SGML, 1986] and later XML [Bray et al., 2004]. Such meta-language oriented documents formats are often called structured documents.

One of the objectives pursued in my research work is to enforce as much as possible the separation principle in order to make content processing more powerful. To that end, my approach consists in designing independent yet combinable languages to describe the document layout, its temporal behavior or synchronization, its network of hyperlinks together with clean programming interfaces giving well-defined access to content and its various facets. The ultimate goal is to achieve a universal content representation that accounts for all of its dimensions while remaining adaptive enough (by transformation) to accommodate its various uses.

2.2 Document dimensions

Structured document models are based on an abstract representation that reflects the internal structure of the documents [ISO/IEC, 1997]. In general, the document structure description is organized around four dimensions [André et al., 1989; Layaïda, 1997], if we except interfaces with programs:

- **The logical dimension** allows to link "syntactically" document elements (or objects) by means of composition or order relations. For example, a chapter may contain sections and an author description can be characterized by a family name followed by one or more given names: first given, the second given, etc. These links are inherently hierarchical and ordered.

- **The spatial dimension** allows to describe style information (such as fonts and colors for text) and geometrical arrangement of elements on screen or paper. It allows also allocating channels and rendering attributes for other types of content such as audio, video and animations.

- **The temporal dimension** allows to describe the temporal organization of objects in the document overs time and synchronization relations between them. Each document element is characterized by a time interval during which it is displayed or heard.

- **The hypermedia dimension** (hypertext when restricted to text documents) often describes non-hierarchical and more semantic relationships between documents or parts of documents. Such relations describe links,

indexes, cross-references, notes, table of contents, etc. They represent the main support for navigation within and between documents.

Each of these dimensions led to a family of dedicated languages. First, at a higher level, SGML, XML DTD [Bray et al., 2004], XML Schemas, Schematron [ISO, 2006a], RelaxNG [Clark & Murata, 2001] or NVDL [ISO, 2006b] have been defined to serve as meta-languages intended to describe the syntax for other languages (for example those describing the different dimensions including programs such as transformations). Most of the languages used to describe the logical dimension such as DocBook [Walsh & Muellner, 1999] or ATA [ATA, 1997] use these XML meta-languages. Both SMIL for the temporal dimension and SVG [Schmitz & Cohen, 2001] for vector graphics use XML also. However, several exceptions exist. Notably, CSS and DSSSL [ISO/IEC, 1996], for the spatial dimension, do have their own syntax while interestingly XSL-FO [Berglund, 2006], serving similar purposes, does not, as does XQuery for programs compared to XSLT. Languages such as XPath, used for queries, XLink and XPointer (both XPath-based describe the hypermedia dimension) are compact languages involving only short expressions and for that reason do not use XML. Overall, syntax used in document representation and processing is not always uniform but the XML syntax remains widely predominant. Figure 2.1 gives an overview of W3C languages and protocols. On top of the http protocol and URIs, XML is considered as one the main metalanguages for the other web languages.

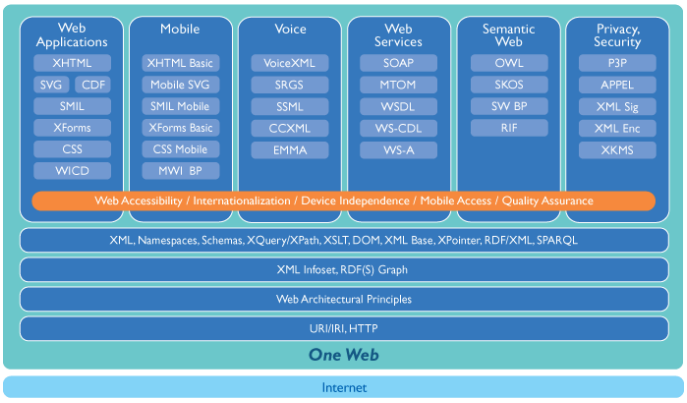


Figure 2.1: W3C languages and protocols stack.

A traditional (static) electronic document is only equipped with the first two dimensions. A hypermedia document (often called hypertext when content is restricted to text) is a static document to which the hypermedia dimension is added. When the temporal dimension is involved, it is often referred to as a multimedia document.

Languages for expressing the formatted version of a document on the web, which is displayed to the user, form a particularly important family of languages. They represent a presentation vocabulary common to all documents

and applications on the web and their support is natively supported once and for all within browsers. As a result, they play a key role and are crucial in the web infrastructure. They represent also the visible part of the web accessible to users and indexed by crawlers of search engines.

In general, document-processing programs are often complex and costly to design. In order to make such programs more reusable, they are designed to operate on document models (grammars) instead of specific instances. Typical examples are XML transformations that produce Web pages from XML documents. When such transformations are designed to work on schemas, all conforming instances can benefit from this processing. Similarly, if pages graphical forms need to be updated, only the style or transformation attached to the model is modified, and updating pages is simply obtained by reapplying this new style/transformation to all documents. For such generic processing to work, transformations and style sheets need to rely solely on the type information contained in the document elements. Type information is mainly carried by tag names and the structural properties (composition and order) of the different elements in the grammar.

In practice, there are two types of language on the web:

User-oriented Languages: Languages like HTML which are intended for display. They are often tolerant to syntactic rules such as incorrect hierarchical nesting of elements and incorrect use of opening and closing tags: **well-formedness**. Most of the browsers do not require syntactic correctness to display the document. This tolerance originates back to the first days of the web where HTML parsers were designed to accommodate the maximum syntax error as the web pages were produced primarily using text editors. Web application designers continue to rely on such browser tolerance and do consider that well-formedness and validity are too cumbersome. Such tolerance has greatly contributed to slowing the adoption of XML in browsers as evidenced by the latest HTML5 where more syntactic rigor is still questioned. The large amount of incorrect documents, HTML-producing applications combined with the successive HTML versions standardized during the last two decades make HTML parsing quite challenging. The HTML parser represents one of the most complex and sensitive pieces of software in browsers and crawlers alike on the web.

Machine-oriented languages: These are the languages based on the XML syntax that require that documents are be both well formed and **valid**. Languages such as BPEL [Jordan & Evdemon, 2007] WSDL [Christensen et al., 2001] or UDDI [Clement et al., 2004] for web services [Gudgin et al., 2007], DocBook [Walsh & Muellner, 1999] documents and formats such as XSL-FO [Berglund, 2006] being mainly manipulated by the programs, incorrect documents may result in errors serious enough to compromise the entire processing outcome.

In the remainder, documents are considered to belong to this second category. Syntactic correctness is important as it represents the main vehicle for introducing generic processing but also relations in the document structures.

At a more abstract level, documents can be considered as entities composed of a finite set of basic pieces of information linked by relationships of different sorts (compositional for the logical dimension, spatial, temporal and hypermedia). These basic pieces are also equipped with properties such types, geometric sizes and durations giving them an existence along each of these dimensions. Documents are themselves organized into classes using models. Models, properties and relationships allow describing a fairly wide range of documents in use today.

2.3 Time-based documents and runtime support

Publications: ACM Multimedia 1998 [Jourdan et al., 1998], DDEP 2000 [Villard et al., 2000].
Students: Loay Sabry-Ismaïl and Lionel Villard, PhD Thesis.

The first years of my research work were dedicated to extending Madeus 1 [Layaïda, 1997], designed during my PhD, to generic document models. The result of this work is called Madeus 2. Madeus 2 refers to a document model and an authoring and presentation system at the same time. The motivation behind the design of Madeus 2 [Villard, 2002; Villard et al., 2000] was to equip user defined XML document models with all of their dimensions and to provide a strong integration between these dimensions. As a first step toward that goal, was the definition of a generic presentation vocabulary for all XML classes where the temporal dimension plays a central role. Then, the general approach I followed consists of producing the presentation by relying on XML types to generate, by transformations, the other dimensions.

One of the major distinctive characteristics of Madeus 2 [Villard, 2002], compared to systems proposed in the literature, is a generalized use of constraints. These constraints are introduced via relations, as a means to describe the entire presentation of a document. For example, the spatial positioning is specified using geometric relationships such as «right of», «left of», «centered with», etc. Temporal synchronization is also performed by means of relations such as «during», «before», «after», «strongly synchronized», etc. These various relations are derived from schema information, conveyed by the element names (their type). However, the coexistence of relations of different dimensions yields complex structures that are often intricate and differ in shape from the logical organization. For example, geometric box nesting used in layout models does not necessarily match nesting used for temporal synchronization nor schema ones. As a result, the authoring process consists of first describing content elements independently of their use in the document. Then, they are inserted into the presentation by reference. This promotes content reuse within the same document without resorting to duplication. In Madeus, relations are added either to named elements (via their identifiers) or generically when they are derived from the type.

A downside of using constraints is the possible introduction of inconsistencies. Madeus 2 implements inconsistency detection algorithms to enforce constraint satisfiablility in documents. In addition, values for the spatial and

temporal positioning such as durations and positions on the screen are automatically calculated by constraint solvers [Villard et al., 2000]. Relations allow to capture more semantic presentational interdependencies within the document and relieves authors from specifying low-level details such as values. The work on Madeus 2 served as the basis for the SMIL language, incremental transformations and automatic adaptations described hereafter.

2.3.1 The SMIL language

Publications: W3C recommendations SMIL 1 [Hoschka et al., 1998], SMIL 2.0 [Layaïda & Ossenbruggen, 2001; Ayars et al., 2001], SMIL 2.1 [Layaïda, 2005; Bulterman et al., 2005], SMIL DOM Note [Schmitz et al., 1999] and CCBR 1998 [Layaïda & Karmouch, 1998].

The goal behind the design of SMIL (Synchronized Multimedia Integration Language) was to promote a universal standard language for temporal synchronization on the web. The main idea was to extend presentation languages such as HTML with synchronization features using markup. A first version was defined in 1998 and the language evolved in successive versions by incorporating more and more features. A typical SMIL presentation contains references (URIs) to media objects but does not include multimedia objects themselves. The core of the language is the temporal model but SMIL describes also other time-dependent features such as animation and a simple form of content adaptation to terminals (choice of content based on language, flow, resolution, etc.). SMIL hypermedia links are also timed and provide time sensitive access to different portions of a presentation. A peculiarity of the language is that it does not describe the encoding format for media objects for transport protocols, but focuses on the other aspects of a multimedia presentation: their synchronization.

Synchronization in SMIL is structured by a nested set of temporal containers delimited by XML tags. Instead of using a single fixed time reference, SMIL timing is made relative using these containers. Time containers allow for displaying elements in parallel (<par> tag), in sequence (<seq> tag) or in a mutually exclusive manner (<excl> tag). If the objects of a container are parallel, specific attributes allow the description of finer synchronization constraints such as lip-syncing a video with an audio. In addition, the temporal model can also describe the user interaction and combining scheduling with interactivity. These interactions are introduced by "events" that can trigger cascades of other events such as the start or stop of other elements in a presentation. In addition, SMIL animations can combine tightly, via interpolations, vector graphics constructs such as geometric paths with timing to yield complex continuous visual effects. The language syntax was designed specifically to integrate well with other host XML languages such as SVG and XHTML via namespaces. Overall, proposed timing constructs and synchronization schemes are actually much richer and can describe even more complex scenarios with relative ease (declarative). In addition, the SMIL language has been designed to be extensible and has several flavors [Layaïda & Ossenbruggen, 2001]: basic,

tiny, mobile, advanced mobile and full together with a scalability framework. A full description of the language is beyond the scope of this document and a gentle introduction to SMIL can be found in [Bulterman, 2001].

I played a key role in SMIL. I was one of the initial founders of the technical group and a main contributor. In particular, I contributed to the definition of the language requirements and edited the technical specification of the recommendation. I have also contributed to reference implementations, a prerequisite for specifications to reach the recommendation status. Today, SMIL is used in mobile telephony as the main format for MMS [3GPP, 2009] (Multimedia Messaging) and for interactive streaming services PSS [3GPP, 2011]. It is also used as a central component in the DAISY language, which is a standard for people with disabilities. It is deployed in a large number of industrial applications such as advertising, high-definition camcorders, audio books, etc. At ISO, SMIL was introduced to serve as a textual scene description for MPEG4 (XMT) [Kim et al., 2000]. Recently, other team members have also studied and implemented SMIL Timesheets [Cazenave et al., 2011]. This proposed standard offers a generic mechanism to add SMIL timing to XML. The work shows explicitly how SMIL Timesheets can be combined with HTML 5 [Hickson, 2011], CSS [Etemad, 2010] and SVG [Dahlström et al., 2011; Schmitz & Cohen, 2001] in a multimedia document.

2.3.2 Synchronized Presentations Scheduling

Publications: TSI 2002 [Hagimont & Layaïda, 2002], MTAP 2002 [Layaïda et al., 2002], SMC 2001 [Layaïda et al., 2001] and [Fargier et al., 1998].
Student: Loay Sabry-Ismaïl, PhD thesis.

If we focus on the temporal dimension, a document presentation is the process of rendering media objects over time according to the temporal scenario specified at authoring stage. The various objects are displayed according to values specified explicitly or calculated from temporal relations. Since content is generally stored in a distributed environment, such as remote audio and video servers, this operation is subject to limited and time varying resources (bandwidth, CPU, etc.). These constraints often result in some temporal deviations and uncertainty. Some objects behave in an indeterministic manner and their durations do not necessarily match the specified values. Remote access latencies of audio and video streams, layers of buffering, congestion are common examples of delay sources. These delays can propagate in the scenario, via temporal relations, affecting the presentation global synchronization and its quality.

It is then necessary to ensure that timing constraints are met as much as possible despite these uncertainties. There are two approaches for dealing with presentation indeterminism. In the first approach, most commonly used, each time the presentation gets out of synchronization, some actions are taken immediately to bring the scenario back to the predefined case. Usually this operation is achieved at the cost of either blocking some of the objects, skipping the content of some objects or delaying some others. In other words, the author

specifications are violated. In the second case, one attempts first to determine if these delays can be taken into account while maintaining or at least remaining close to the scenario specified by the author. The later method requires some means to adjust the scenario according to the author's intent.

The general idea I explored belongs to the second method. It consists of taking advantage of the flexibility of temporal scenarios to build synchronization algorithms that incur the least de-synchronizations in the scheduling operation. First, I have investigated reasoning techniques for analyzing temporal constraints under uncertainty [Fargier et al., 1998]. Second, I proposed the use of intelligent schedulers, which allows rendering of a multimedia presentation while handling the indeterministic behavior of objects. These schedulers monitor the evolution of a scenario in response to different events of a presentation (start event, termination, notification of delays). The elements of a multimedia presentation are modeled with two basic constructs: controllable and incontrollable temporal intervals. These two constructs distinguish interval durations that can be adjusted by the scheduler from those corresponding to external uncertainties. The overall presentation scheduling is then captured by a directed acyclic graph where nodes represent instants or events and arcs represent temporal intervals differentiated according to their type. The flexibility of controllable objects in the graph is then used to compensate for unpredictable delays of incontrollable objects according to the graph topology. When a solution becomes impossible to find, the scheduler minimizes the impact of desynchronization in space (number of violated requirements) and time (desynchronization period). This work was conducted in collaboration with Verimag Laboratory [Altisen et al., 1999]. Furthermore, Gregor Gößler devoted a detailed study of this problem in his master's thesis [Gößler, 1998] by exploring automata-centric methods. More recent studies continue to improve on these results in particular in the framework of controller synthesis [Abdeddaim et al., 2009].

As a complement to client-side synchronization, I studied end-to-end system methods to improve the overall quality of service of a presentation. In particular, I explored code mobility as a means to dynamically adapt remote content to fit client timing constraints and resources. Mobile code has been also used to extend the functionality of a remote content server on behalf of a scheduler. For example, such code can adjust audio and video data prior to their transmission to the client application. A proxy-site is also an interesting peer to achieve this task since typical mobile-oriented infrastructures are often split in two levels of performance. The first level corresponds to fast content and application servers interconnected with reliable backbone connections and the second level is composed of performance varying devices with wireless connectivity. The experimental results we have obtained show that it is possible to gracefully adjust content access to client computational power and to available bandwidth. We have also shown that such infrastructure can be used to adapt statically encoded streaming content and personalize parameters such as size, resolution, number of colors, etc. In addition, the code deployed remotely is

often very negligible in size compared to video and audio payloads. The overall interoperability can also be improved as it becomes possible to adapt on the fly formats and protocols to client supported ones [Hagimont & Layaïda, 2002].

2.4 Document Adaptation

The means of accessing the web have greatly evolved in recent years by the introduction of both various types of terminals and communication media channels. This period has witnessed an impressive growth in wireless broadband networks deployment that became an integral part of the Internet. Both mobile phone networks such as third generation networks and WiFi are now part of the web infrastructure. At the same time, more and more devices became connected such as cellular phones, tablets, television sets, game consoles and all sorts of embedded devices. All are multimedia, in the sense that they are used to access multiple media, including rich web content and continuous media such as video and music.

This double evolution in content and access methods raises the question of the adequacy between content and the means for accessing it. With this variety, one needs to ensure an access in good conditions to the same multimedia resources, from different devices, using different software, and across networks with different characteristics. Yet the vast majority of content deployed on the web was designed for desktop PCs, which have generally more resources than other terminals. This is the problem of content adaptation that we address in this work.

The pragmatic approach to adaptation consists in developing content variants targeted to particular devices, possibly in a specifically tuned format or representation. This is typically the approach used by many content publishers: a website for each terminal. It is also similar to choices made in the past by telecom operators such as WAP [OMA, 2001a]: the WML format [OMA, 2001b], optimized for small screen terminals with low bandwidth connectivity. But these approaches contradict the web philosophy of building a universal information space accessible to all. The risk they introduce is web fragmentation that would lead to a web for mobile phones containing only the information produced for these devices, a web for TVs, a web for PDAs, etc. This brings strong user restrictions for each type of device: they have access to only part of the information. This also leads to high costs for content producers seeking a wide audience: they must produce several times the same information in different formats for different types of devices.

In practice, designers produce a very limited set of such adapted content: usually, one for desktop PCs and a second one for mobile terminals. The reason behind this is that they are more comfortable with specific designs tied to particular screen sizes and terminal characteristics. When they create content, they do generally engage in a cyclic process where they specify and check the result directly on a specific terminal. With such a process, it becomes hard to address correctly a richer set of terminals. Furthermore, multimedia documents

formats are now defined as progressively richer sets of sub-languages with increasing features. This will likely make any adaptive design more complex if not impossible without automatic processing.

The work I have carried out consists in developing automatic adaptation methods for multimedia content together with supporting architectural frameworks. More precisely, I proposed two adaptation approaches; one based on structural transformations operating on a more advanced web architecture, the other based on presentation "semantics".

2.4.1 Adaptation by structural transformations

Publications: TSI 2005 [Layaïda et al., 2005], MDM 2004 [Lemlouma & Layaïda, 2004], SAINT 2003 [Lemlouma & Layaïda, 2003a], ELPUB 2003 [Lemlouma & Layaïda, 2003b], MMM 2001 [Lemlouma & Layaïda, 2001].
Student: Tayeb Lemlouma, PhD thesis.

The first approach we considered promotes the use of as "universal" as possible formats. In this approach, each document is authored only once and stored in a single form. When a client accesses the document, an adapted version that takes into account the specific client "context" is generated, by transformation, from this single source. The context includes the hardware and software characteristics of the client device, its network capabilities, but also user abilities/disabilities and preferences.

First, we have defined a general architecture of automatic content adaptation (see Figure 2.2) called NAC. Then, we studied how to modularize web languages to create progressively richer sets of sub-languages with increasing features. At a syntactic level, modularization consists in describing languages in a more flexible way by combining XML elements and attributes into functional groups. Each group is used to describe a particular aspect of the document, such as layout, synchronization, transitions, animations, etc. A group can in turn be organized as a set of modules that correspond to functionality levels from the most basic to the more advanced. This creates dependencies between modules.

A language for a given device or conforming to a certain profile is described as a consistent collection of modules. This collection defines an XML language corresponding to the target of the adaptation transformation. When a web document is accessed, a negotiation phase between the client and the server is engaged to identify this target language. Since documents are also encoded using a particular profile (called document profile), adapting falls back to transforming such content to the negotiated one. Transformations have been modularized in NAC so that they are capable of transcoding between any pair of SMIL language profiles.

To facilitate this negotiation phase, a general schema, called UPS (Universal Profiling Schema) was defined for describing machine capabilities (terminals, servers or proxies), users preferences together with network characteristics. UPS is built on top of CC/PP [Klyne et al., 2004] and uses RDF [Lassila & Swick, 1999]. This schema also covers content item descriptions such as

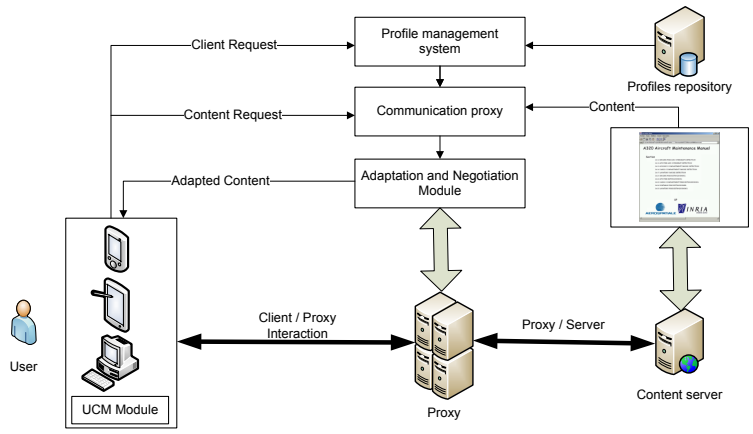


Figure 2.2: Automatic Content Adaptation Architecture

basic media elements (audio, video, etc.) and allows for specifying equivalence relations between objects. Equivalence relations include also media conversion operations, such as image transcoding between different formats and modalities such as text to speech. The results of this work have been contributed to CC/PP [Klyne et al., 2004] and Device Independent Authoring [Smith, 2010] W3C working groups.

Today, web content servers offer very limited support for negotiated content. The HTTP protocol supports only some basic negotiation based on language, media types and character encoding [Holtman & Mutz, 1998]. But usually content structure is served as is to all clients. With increasing diversity on the web, we foresee that the current infrastructure will evolve toward more content-centered adaptation. The transformations explored in this work are merely syntactic and sometimes lead to insufficient quality adaptations. To enhance this quality, I explored a more "semantic" approach.

2.4.2 Semantic adaptation

Publications: MTAP 2011 [Laborie et al., 2011], IJCAI 2003 [Euzenat et al., 2003].

Student: Sébastien Laborie PhD thesis, in collaboration with Jérôme Euzenat.

In the second line of work on adaptation, I proposed a semantic framework in which a multimedia document is interpreted as a set of potential presentations [Euzenat et al., 2003]. The framework defines adaptation as the search for a document compliant with the device constraints, which is as close as possible to the presentations specified originally by an author. It does it in a very general way, independent from the concrete multimedia document languages and independent from the main multimedia document dimensions, i.e., temporal, spatial and hypermedia. In [Laborie et al., 2011], we show how this approach can be used for adapting the structure of multimedia documents in concrete multimedia description languages without having to anticipate the type of constraints imposed by the devices.

For that purpose, we define a relational structure that captures the spatio-temporal and hypermedia dimensions of multimedia documents. We have developed an adaptation algorithm, which transforms in a minimal way such a structure according to device constraints.

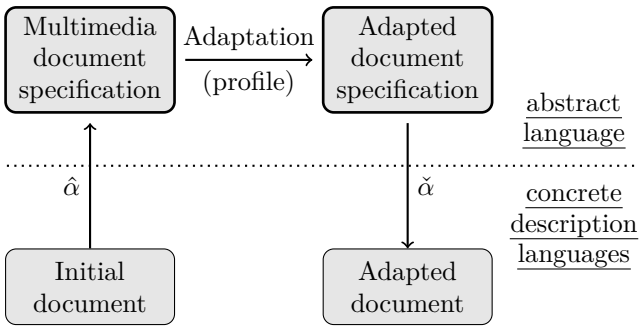


Figure 2.3: Multimedia document adaptation strategy: a document in a concrete language is first ($\hat{\alpha}$) expressed as an abstract specification on which adaptation is performed; then, the result is transformed back in the initial format ($\check{\alpha}$).

In order to capture document dependencies semantically, we rely on an abstract relation graph. From this initial relation graph and a given profile, we compute adapted relation graph solutions, which are, close to the initial one and such that all of their relations satisfy the profile. In particular, we consider all possible relation graphs that satisfy a given profile and for each of them we select those that are at minimal distance from the initial relation graph. The distance between two relation graphs depends on a proximity measure between relations beared by the same arc in both graphs. We consider that the proximity between two relations relies on the conceptual neighborhood between these relations and is measured by the shortest path distance in the corresponding neighborhood graph.

The resulting semantic adaption is a three-step process: (1) abstracting the original content format, followed by (2) adapting that abstract representation to the device profile, and (3) producing an adapted version in the same format as the input document (see Figure 2.3). This approach has many advantages. First, it allows for defining a device independent format in the form of some abstract relations between multimedia document objects; as new format appears, it can be used by only defining rules for extracting the abstract format from the concrete one ($\hat{\alpha}$) and vice-versa ($\check{\alpha}$).

2.5 Transformation-based authoring

Publications: WWW 2002 [Villard & Layaïda, 2002], DDEP 2000 [Villard et al., 2000].
Student: Lionel Villard, PhD thesis.

One of the main obstacles in using structured multimedia document models is the lack of adequate XML authoring methods and tools. It is necessary to have the ability to produce content easily but also to design transformations capable of generating its displayed form, i.e. the presentation. These transformations are crucial since they are also used on content servers to produce such presentations at access time. However, XML transformations are very often associated with a "batch" type of design methods, execution and testing. As a result, the transformation creation process is a tedious one in which programers update, enrich, then empirically verify for correctness by applying some test content. This kind of design method has several disadvantages:

- The heaviness of the design process: any change in the transformation leads to a new cycle,
- The difficulty of identifying the sources of errors and to assess the effect of a change on the transformation result,
- The cost of document presentation production at authoring: a small modification of the source content requires reapplying the whole transformation.

A typical operation that does not accommodate these drawbacks is interactive transformation-based content authoring. One of the key aspects for the success of such frameworks is related to performance. In order to be usable, they must be fast enough and context driven to meet the needs of WYSIWG editing. In particular, it is critical that modifications of the source document or the transformations are reflected promptly to the user. Therefore, making transformation programs incremental becomes a major issue. Incremental changes allow controlling the scope of the document changes so that they do not require a global re-evaluation of the entire transformation, a costly operation.

In order to design such incremental methods, we have studied how to turn the popular XSLT transformation language to become incremental. The general idea consists of examining transformations in order to isolate, for each modification type (content and transformation instructions), the document portions that need to be re-evaluated. More precisely, we perform some static analysis of XPath expressions, corresponding to patterns in XSLT, to identify these portions. This analysis determines accurately the list of instructions to be re-evaluated (called re-evaluation rules in our work). When a source node matches the path expression, then the list of associated instructions becomes likely to be re-executed. This analysis is facilitated by the pure nature of the XSLT language, which is side effect free. We built an incremental XSLT evaluator, which was used as a core engine for DocBook [Walsh & Muellner, 1999] and ATA [ATA, 1997] authoring tools. For example, the ATA tool was used to showcase the editing of aircraft technical documentation (Lionel Villard's thesis was carried out in partnership with Airbus [Villard, 2002]).

The lessons learned from this work show that obtaining efficient incremental transformations depends on how transformations are written. For example,

limiting recursion and wildcard steps in path expressions can greatly enhance performance. Another optimization is also related to path expressions. In order to identify the list of instructions to re-evaluate, a pattern matching is performed on the entire re-evaluation rule set. To avoid such an overhead, one needs to establish statically the containment relations between all paths contained in these rules. A pattern $p1$ is contained in another pattern $p2$ when all nodes that match pattern $p2$ match pattern $p1$. So, in order to find the list of instructions to re-evaluate, a successful pattern matching against $p1$ makes pattern matching against $p2$ useless.

In general terms, more precise path expressions analysis will lead to more accurate selections, and subsequently diminish useless re-evaluations and enhance performance. For that purpose, a more systematic and finer grained analysis of path expressions combined with schemas becomes necessary.

2.6 Types and paths analysis

Publications: PLDI 2007 [Genevès et al., 2007a], TOIS 2006 [Genevès & Layaïda, 2006b], DKE 2007 [Genevès & Layaïda, 2007], IJCAI 2011 [Bárcenas et al., 2011], WWW 2012 [Genevès et al., 2012], Extreme Markup 2003 [Vion-Dury & Layaïda, 2003].
Students: Pierre Genevès and Everardo Bárcenas Patiño, PhD thesis.

In the previous sections, we have emphasized the central role played by transformations in XML processing. They are widely used to produce presentations in content servers and to achieve all sorts of tasks. XML applications most commonly use schemas for performing validation (also called *dynamic type-checking*). Validation consists in using a schema validator that analyzes a particular XML document in order to ensure that the document actually complies with the schema. However, XML documents are often generated dynamically. Typically, programs that manipulate XML first access data (possibly conforming to an available schema) using XPath expressions, and then combine the results and output XML documents expected to conform to a given schema. In addition, in the current web architectures increasingly dominated by web services, content often undergoes several transformations before producing a result. In addition, XML applications do frequently involve large volumes of content. For example, in technical documentation, a single maintenance manual for the A320 Airbus studied in [Villard, 2002] weights several hundred of megabytes. Consequently, it becomes crucial to analyze the XML transformations in order to make them safe and efficient.

One of the key safety properties is to guarantee that transformations produce valid documents against their schema [Akpotsui, 1993; Akpotsui & Quint, 1992]. This is called *static type-checking* and consists in ensuring at compile-time that invalid documents can never arise as outputs of XML processing code. A static type checker analyzes a program, possibly in conjunction with schemas that describe its input and output (depending whether such schemas are available). This problem is known to be difficult. The static analysis of the complete XPath language alone is undecidable. The variety and wide

use of XPath in XML applications nevertheless raise some important research questions: what is the largest XPath fragment with decidable static analysis? Which fragments can be effectively decided in practice? How to determine if an XPath expression is satisfiable on any of the XML trees defined by a given schema? Does the result of an XPath expression over a valid document always conform to another schema? Is there an algorithm able to answer these questions in an efficient way so that it can be used in practice?

One source of difficulty in such questions is that it involves a possibly infinite quantification over a set of trees. XML types denote such sets. Furthermore, a variety of factors contribute to its complexity such as the operators allowed in XPath queries and the combination of them (recursion, bidirectional navigation in trees, qualifiers, etc). On the other hand, schema languages have been extensively studied and are now well understood as subsets of regular tree languages [Murata et al., 2005]. However, although many attempts have been made for better understanding static type-checking techniques, in particular through the design of domain specific languages [Hosoya & Pierce, 2003; Benzaken et al., 2003], dealing with XPath effectively remained unsuccessful.

We started from the idea that two issues need to be answered in order to solve such decision problems in XML. First, it was essential to identify an appropriate formal framework with sufficient expressive power to capture both regular tree languages and navigation introduced by XPath. Then we had to find appropriate techniques to solve the satisfiability problem in that framework. satisfiability allows determining whether a given property holds or not. Then, it would be interesting to obtain an XML document that exemplifies it. Such properties include the XPath containment, emptiness, equivalence and coverage of XPath queries (in the presence or absence of regular types of trees).

A first important result was achieved through the design of a finite tree logic adapted to XML, and its decision procedure. The logic is expressive enough to capture regular tree types along with multi-directional navigation in finite trees. It is decidable in single exponential time (specifically in $2^{O(n)}$ steps where n is the size of the input formula defined as its number of atomic propositions and eventualities). This improves the best-known computational complexity for finite trees. Another contribution is that we showed how to linearly compile queries and regular tree types (including DTDs and XML Schemas) in the logic. This offers a uniform notation for both constructs and facilitates reasoning on them. The logic enjoys the nice property of being closed under boolean operations which allows for reducing many problems to satisfiability. It supports the full navigational features of XPath and covers the largest fragment considered in the literature [Marx, 2004]. From the algorithmic point of view, the decision procedure proved entirely feasible using symbolic techniques borrowed from verification: BDD (Binary Decision Diagrams).

2.7 Managing evolution

Publications: ICFP 2009 [Genevès et al., 2009], TOIT 2011 [Genevès et al., 2011], WWW 2010 [Layaïda & Genevès, 2010], ICSE 2011 [Genevès & Layaïda,

2011].

Collaboration: Vincent Quint and Pierre Genevès.

In this work, we consider the problem of XML Schema evolution. In the ever-changing context of the web, XML schemas continuously change in order to cope with the natural evolution of entities they describe. Schema changes have important consequences. First, existing documents valid with respect to the original schema are no longer guaranteed to fulfill the constraints described by the evolved schema. Second, the evolution also impacts programs manipulating documents whose structure is described by the original schema.

We propose a unifying framework for determining the effects of XML Schema evolution both on the validity of documents and on queries. The system is very powerful in analyzing various scenarios in which forward/backward compatibility of schemas is broken, and in which the result of a query may not be anymore what was expected. Specifically, the system offers a predicate language, which allows one to formulate properties related to schema evolution such as compatibility. The system then relies on exact reasoning techniques to perform a fine-grained analysis of programs and schema changes. This yields either a formal proof of the property or a counter-example that can be used for debugging purposes. The system has been fully implemented and tested with real-world use cases, in particular with the main standard document formats used on the web, as defined by W3C. The system identifies precisely compatibility relations between document formats. In case these relations do not hold, the system can identify queries that must be reformulated in order to produce the expected results across successive schema versions. The long-term goal of this work is to find methods and techniques to reformulate XML transformations automatically when schemas evolve.

2.8 Functions, polymorphism and subtyping

Publications: ICFP 2011 [Gesbert et al., 2011].

Collaboration: Pierre Genevès and Nils Gesbert (Postdoc).

The growing popularity of programming languages such as XQuery leads to new needs. The first of these needs is the ability to support modularity in larger applications. As applications become more and more distributed, a more adequate support for web services interactions is also necessary. These new needs require polymorphic type systems for XML programming languages. For example, in XQuery [Boag et al., 2006] which is a typed functional language, the support for higher-order functions, currently absent from the language, appears as a requirement in the forthcoming third version [Engovatov & Robie, 2010]. Higher-order functions and parametric polymorphism are two of the most powerful constructs in functional programming languages such as ML, Caml or Haskell. In these languages, functions are considered as values like any other values such as integers, lists, etc. In addition, they can be passed as parameters or returned as results.

In XML, it is attractive to reach such powerful type systems where types can denote not only data types such as schemas but also computations. To

that end, function types need first to be supported in the manner of [Benzaken et al., 2003; Castagna & Xu, 2011]. In addition, if functions can be made parametric using type variables (parametric types), they become more generic since they can operate on a large number of specific types. Such functions are also important to promote code reuse.

We have studied parametric polymorphism for type systems aiming at maintaining full static type-safety of functional programs that manipulate linked structures such as trees, potentially with higher-order functions. We consider a type algebra equipped with recursive, product, function (arrow), intersection, union, and complement types. We first have shown how the subtyping relation between such type expressions can be decided through a logical approach.

Our main result solves an open problem: we prove the decidability of the subtyping relation when this type algebra is extended with type variables. This provides a powerful polymorphic type system (using ML-style prenex polymorphism, where variables are implicitly universally quantified at top level), for which defining the subtyping relation is not obvious, as pointed out in [Castagna & Xu, 2011], and for which no candidate definition of subtyping had been proved decidable before. The novelty, originality and strength of our solution reside in introducing a logical modeling for the semantic subtyping framework. Specifically, we model semantic subtyping in the finite tree logic presented earlier and rely on a slightly modified satisfiability solver in order to decide subtyping in practice. We obtain an EXPTIME ($2^{O(n)}$) complexity bound as well as an efficient implementation in practice.

2.9 Research Summary

The guiding motivation of my work is to promote declarative and typed representation of content in documents and web applications. The overall goal is to ease the design of web applications and make them richer, safer and more efficient. During the last few years, I focused first on enriching content representation to support a wider and more integrated set of features: temporal synchronization, spatial positioning, logical organization and hypermedia links. The central idea of my work consists in defining the various document dimensions or facets with well-defined languages. Then the idea is to make them combinable, by composition, within general document models such as Madeus or more generally XML-defined languages. When possible, my work promotes standard and universal languages. My work on temporal synchronization resulted in a W3C recommended language called SMIL. This language has been used in mobile infrastructure for Multimedia Messaging and integrated in other languages such as SVG.

By making the different content facets more explicit, declarative languages make possible advanced content processing such as intelligent scheduling, incremental manipulations, automatic adaptation and personalization, verification of content manipulations in programs, code optimizations, etc. Furthermore, dedicated languages such as XQuery are becoming more mature and offer

better-suited programming frameworks. They put back regular tree types together with path expressions as first-class constructions in the language and the corresponding type system. They also offer a programming model, which can potentially be deployed at the three tiers of web applications (browser, middleware and storage). Recent studies have shown substantial gains in terms of code readability and size [Bamford et al., 2009]. In addition, there is now a real potential to build even more advanced tools based on these languages. XQuery, for example, is equipped with a type system [Draper et al., 2010]. It becomes possible to use static analysis techniques to enforce type safety and optimize programs automatically.

Content processing that I studied in priority was related to typical content manipulation tasks such as authoring, transformations, presentation rendering and scheduling, etc. My work on incremental evaluation and adaptation shows that it is possible to obtain advanced features by concentrating on analysis:

- content analyses to equip documents with automatic adaptation capabilities
- transformations analyses to provide advanced editing tools based on incremental evaluation or static type verification.

One of the major challenges for XML program analysis is to model appropriately types and queries, as they are the main constructs to model contents and operations on them. Theses two constructs are also fundamental theoretically. XPath regular path queries, when slightly extended [Marx, 2004], exactly correspond to first order logic on finite tree structures with two free variables. On the other hand, XML schemas correspond to monadic second order logic (MSO) on theses structures [Rabin, 1969].

We have shown that it is possible to obtain a logic covering these two powerful constructs uniformly while remaining effective in practice. The difficulty that we overcame is both theoretical and practical. We had to first establish a precise complexity bound for reasoning on such constructs. Then we had to find appropriate techniques to solve them. In particular, the symbolic techniques that we have introduced proved effective to achieve this dual objective. More recently, we were also able to extend the logic to support more directly XML computation constructs such as functions, higher-order function and polymorphism. As a result, we have been able to extend the analysis to richer type algebra. This highlights subtle interconnections between logic on one side, programming languages on the other side, which can be combined in advanced tools such as compilers and other analyzers. This opens the door for more ambitious and unified end-to-end programming models covering content description, processing and distribution. At a higher level of abstraction, applications can be seen as a set of distributed functions connected by web services. They can be analyzed for non-functional aspects such as performance by code and data distribution, security and privacy enforcement while enhancing their scalability on increasingly popular infrastructures such as the cloud.

Three

Curriculum

3.1 Supervised PhD students

For each student, I describe here the thesis research subject, the defense date, the current student position and, when applicable, co-supervisors names. The present list corresponds only to students to whom I served as main supervisor and whose research subjects where those I was in charge of as principal investigator.

Loay Sabry-Ismaïl: Distributed execution schema for multimedia documents. PhD thesis. Université Joseph Fourier. 25 January 1999. Co-supervised with Vincent Quint.
Assistant Professor at Qatar University, Doha, Qatar.

Lionel Villard: Document models for authoring and adapting multimedia presentations. PhD thesis. Institut National Polytechnique de Grenoble. 21 March 2002. Co-supervised with Cécile Roisin.
Permanent researcher at IBM Watson Research Center, New-York, United-States.

Tayeb Lemlouma: Multimedia Services Negotiation and Adaptation Architecture in Heterogeneous Environments. PhD thesis. Institut National Polytechnique de Grenoble. 9 June 2004. Co-supervised with Cécile Roisin.
Assistant Professor at IUT of Lannion (Institut Universitaire de Technologie-Université de Rennes I), France.

Pierre Genevès: Logics for XML. PhD thesis. Institut National Polytechnique de Grenoble. 4 December 2006. Co-supervised with Vincent Quint.
Winner of the EADS prize in Information Sciences, awarded by the European Aeronautic Defense and Space Company (EADS), 2007. Best thesis award of Institut National Polytechnique de Grenoble, France, 2008. Finalist Cor Baayen PhD award, ERCIM (European Research Consortium for Informatics and Mathematics), 2008.
Research scientist at CNRS, France.

Sébastien Laborie: Semantic Adaptation of Multimedia Documents. PhD thesis. Université Joseph Fourier. 28 May 2008. Co-supervised with Jérôme Euzenat.
Assistant Professor at IUT de Bayonne et du Pays Basque. Université de Pau et des Pays de l'Adour, France.

Everardo Bárcenas Patiño: Automated Reasoning on Trees with Cardinality Contraints. PhD thesis. Institut National Polytechnique de Grenoble.

14 February 2011. Co-supervised with Vincent Quint.
Postdoc at Rice University, United States.

Melisachew Wudage Chekol: Regular Graph Queries Reasoning. PhD student. Institut National Polytechnique de Grenoble. Third year. 2012. Co-supervised with Jérôme Euzenat.

Postdocs:

Nils Gesbert, Tayeb Lemlouma.

Engineers:

Romain Deltour, Julien Guyard, Jan Mikáč, Alain Uginet, Yves Carbonneaux, Daniel Weck, Peter Hewat.

Masters students:

Loay Sabry-Ismaïl, Maximilien Laforge, Laurent Garçon, Stéphane Martin, Sébastien Chassande-Barrioz, Jean-Charles San Severino, Daniel Weck, Peter Hewat, Sébastien Laborie, Victor Diaz, Nebil Ben Mabrouk, Imène Issaoui.

3.2 Projects and Funding

ANR project Typex (2012-2015):

Partners: Preuves, Programmes et Systèmes (PPS) Laboratory, Université Denis Diderot. Laboratoire de Recherche en Informatique (LRI), Université Paris-Sud.

The goal of this project is to produce a new generation of XML programming languages stemming from the synergy of integrating different approaches into a unique framework. Languages whose constructions are inspired by the latest results in the programming languages research; with precise and polymorphic type systems that merge PL typing techniques with logical-solver-based type inference; with efficient implementations issued by latest researches on tree automata and formally certified by latest theorem prover technologies; with optimizations directly issued from their types systems and the logical formalizations and whose efficiency will be formally guaranteed; with the capacity to specify and formally verify invariants, business rules, and data integrity, languages with a direct and immediate impact on standardization processes.

ANR project Codex (2009-2012):

Partners: Gemo EPI, Inria Saclay, Mostrare EPI, Inria Lille, LRI Laboratory, Université de Paris Sud, PPS Laboratory, Université Denis Diderot, Laboratoire Informatique (LI), Université François Rabelais de Tours, Innovimax Company.

A large family of standards have been or are currently developed by international standardization bodies such as the World Wide Web Consortium (W3C), ISO etc., for tasks as varied as querying, formatting, encoding scientific content, modeling interactions between distributed parties or coordinating such interactions, validating data according to given types, updating, scripting etc. A unique dimension of the XML technology boom is the availability of numerous free tools, enlarging the pool of potential developers, as it has never been before, and fostering experimentation and innovation. Finally, another

unique dimension is the rich interaction between academic research and industrial players, taking place within standardization bodies, in international technical meetings, and more generally in the worldwide electronic arena of those interested in XML technologies. The research work proposed in this project pushed the frontier of XML technology innovation in three interconnected directions. First, we studied languages, algorithms, and developed prototypes for efficient and expressive XML processing, in particular advancing towards massively distributed XML repositories. Second, we considered models for describing, controlling, and reacting to the dynamic behavior of XML corporas and XML schemas with time. Third, we proposed theories, models and prototypes for composing XML programs for richer interactions, and XML schemas into rich, expressive, yet formally grounded type descriptions.

RIAM project SATIN (2003-2005): Satin (Synchronized applications for interactive digital television) is a joint project with the httv company, funded by the French ministry of industry through the RIAM network. Most interactive television services broadcasted today are completely independent from the audio-visual streams, as if the two worlds of television and interactive applications were only sharing the same broadcast channel. The main goal of Satin was to introduce some synchronization between the two kinds of contents. To achieve this goal, a comprehensive environment was designed and implemented for creating, producing, broadcasting and presenting interactive digital television applications that are synchronized with audio-visual contents. This project was based on Web standards, especially formats created by W3C (XML, SMIL, XSLT, etc.), and the digital television standards from the MPEG group. My main contributions concerned formats (based on XML and SMIL), the editing environment, and the simulation component, which allows authors to immediately get feedback on user experience.

Alcatel-Bell (2001-2005): This project was funded by Alcatel-Bell, a large consumer electronics maker company. The collaboration subject was related to the creation and deployment of interactive content on heterogeneous devices such as advanced Internet connected phones, WAP-based mobiles, kiosks, etc. We explored device independent content representation and context-aware adaptation methods for these devices (Device Independent Authoring). We have also explored how to use automatic negotiation and adaptation methods to make content fit the best device capabilities and user preferences. Alcatel specific vocabularies were designed for the company's product lines and an internal content representation called MHML was used as a target language for adaptation. Adaptation process was also used to generate content in other formats such as XHTML, WML and VoiceXML.

Microsoft Research (2002-2004): This project proposal received a Microsoft Research Innovation Excellence Awards for Embedded Systems. The proposed work was related to dynamic adaptation of embedded multimedia applications using Microsoft's Windows CE .Net. Multimedia applications are increasingly deployed and run on mobile terminals, characterized by limited capacities in terms of network bandwidth, CPU, memory or display size. There-

fore, managing Quality of Service for multimedia applications executed on mobile terminals was the main subject of investigation. We have experimented component-based dynamic adaptations on proxy nodes for adapting multimedia streams according to the requirements of the terminals. A perspective to this work is to enable such adaptations on the terminal, which would allow new QoS management strategies. We used Windows CE .Net, compact .Net and DirectShow technologies to implement dynamic adaptation of embedded multimedia applications.

Airbus (1999-2002): This was a joint project with Airbus France (Document Engineering Associated Laboratory). The project was related to the design of generic multimedia document models for aircraft maintenance and the associated tool chains. The goal was to study the feasibility of extending document models such as ATA to multimedia. The second goal was presentation generation using transformations applied to the Airbus A320 maintenance manuals. One of the key aspects was to adapt the timing model in order to coordinate on the field maintenance tasks with document presentation.

Innovatel-Cegetel (2000-2002): This project, funded by Innovatel-Cegetel, aimed at designing a signaling and multimedia messaging system for embedded devices equipped with GPRS. The signaling part was one of the first embedded implementations of the SIP (Session Initiation Protocol) protocol under development at IETF. The explored idea was to enhance messaging by multimedia content via MMS (Multimedia Messaging Service) on the first generation of Smart phones. We have developed an experimental SMIL-Basic client capable of handling MMS but also streaming content compliant with the PSS draft (Packet Switched Streaming Service).

Alcatel (1998-2001): This project was born out of an informal collaboration with Alcatel's Unité Information et Réseau of Corporate Research. The contract was related to the specification, the design and validation of a multimedia language called MHML, Alcatel proprietary language for embedded devices. This work was based on a toolbox built on the Madeus 2 system. The validation part relied on constraint solvers contained in the toolbox, which were adapted to the MHML event-based timing model.

Dassault Aviation (1998-2000): This project, from the Génie program of the French ministry of research, was related to interactive structured technical documentation. The project involved the Opéra team and Dassault-Aviation. My participation to the program was on the following lines: the study of transformation techniques for presentation generation, the extension of structured content models to timing and synchronization and the design of prototypes for compositional multimedia presentations.

3.3 Administrative responsibilities

Member of Inria commission for scientific employment: 2007-to date. The Inria Grenoble Rhône-Alpes commission is in charge of selecting postdoc proposals and candidates, as well as assistant professors and professors delegations and secondments.

LIG laboratory council member (substitute): 2007-2010.
I served as a member of LIG Laboratory Council as a substitute. I was appointed by Brigitte Plateau, head of the LIG Laboratory (Laboratoire d'Informatique de Grenoble).

External project evaluator: 2003-2009.
I am a regular evaluator for several funding organizations such as ACI Masse de Données, RNTL, RNRT, ANR, Qatar National Research Foundation (QNRF), Programmes de coopération scientifique internationale argentina- Inria/CNRS (SECYT), Inria associated teams.

Tenure position referee: 2008.
Referee for tenure position for Computer Science Department at the Worcester Polytechnic Institute, MA, United-States, 2008.

Member of Inria GPCCMI committee: 2004-2007.
GPCCMI stands for «Groupe de Pilotage du Comité de Coordination des Moyens Informatiques». The role of the committee was the management at the national level of budget, support activities organization, project planning, technical assessment of IT services. The committee included five members: Michel Cosnard (research centers directors representative), Christine d'Argouges (head of human resources) and Eric Gautrin (Head of IT Infrastructure) and myself. I have also participated in a followup committee COSS (Comité d'orientations stratégiques et de suivi) whose role was the specification of «schéma d'orientation du système d'information (SOSI)». I was appointed by Gilles Kahn, Chairman and CEO of Inria. 2004-2006.

Inria development commission member: 2002-2005.
The goal of this commission was to propose new development standards and organization for software activities at Inria. We produced collectively a software development best practices charter document entitled: «Un processus de développement logiciel pour Inria». I was appointed by Francois Rouaix, Inria Development Director.

Membre of UJF hiring committee: 2002-2004.
Member of hiring commission (commission de spécialistes) of assistant professors at Université Joseph Fourier, Grenoble.

Inria ISDN representative: 2001-2004.
ISDN (Institut des Sciences du Document Numérique de la région Rhône-Alpes) was a virtual laboratory funded by CNRS in the Rhône-Alpes region coordinating research activities on document engineering.

Scientific manager of Calliope: 2000-2003.
Scientific Manager of the Digital Library Calliope at Inria (developed in collaboration with IMAG and XRCE Laboratories). Calliope was an on-demand scanning system of scientific articles, summary management

servicing Inria researchers. I was appointed to this position by Jean-Pierre Verjus, Inria Grenoble Rhône-Alpes Director.

External scientific evaluator at XRCE: 2002.
External Scientific Evaluator of Document Models and Transformation Technologies Group DMTT at Xerox Research Center Europe, XRCE, Grenoble, 2002.

Engineers hiring committee member: 2000-2002.
Member of Inria Grenoble Rhône-Alpes committe for the employment of Software Development Engineers (Ingénieurs Associés). Appointed by Bernard Espiau, Inria Grenoble Rhône-Alpes Director.

Ideas Laboratory Inria representative: 2000-2001.
I worked one day a week, for a period of one year and a half, at CEA center in Ideas Laboratory (Pôle Minalogic). The project I was involved in was related to the design and implementation of multimedia infrastructure for mobile phones and handheld devices equipped with Leti/CEA sensor prototypes. Appointed by Bernard Espiau, Inria Grenoble Rhône-Alpes Director.

Thesis referee: Examiner
Cyril Concolato. Multimedia scenes description: representations and optimizations. PhD thesis in computer science and networking, Département Traitement du Signal et Images, Télécom ParisTech, Paris. 2007.

Kimiaei Asadi Mariam. Multimedia content adaptation with MPEG-21: resource conversion and semantic scene adaptation. PhD thesis in computer science and networking, ENST-COMELEC Communication et Electronique, ENST. 2005.

Walid Mahdi. Semantic macro segmentation of audiovisual documents using spatio-temporal indices. PhD in computer science, École Centrale de Lyon, 18 Mai 2001.

3.4 Conference organization and program committees

SMIL'2003: European Conference on Synchronized Multimedia Integration Language. I served as the conference organizer and program committee chairman, February 2003, CNAM, Paris.

PDMS'2001: Authoring and broadcasting of synchronized content over the internet (Production et Diffusion sur l'internet de documents Multimédias Synchronisés). Four days summer school. I initiated and organized this summer school and proposed the final program. This school attracted 90 participants: 35 industry, 45 academia and10 invited experts, August, 2001, Autrans, Isère.

W3C WG Meeting'2000: Synchronized Multimedia Working Group Meeting, World Wide Web Consortium, 34 participants, Montbonnot, France, June 2000.

Program committees: I served as a program committee member of the following conferences: Medi 2012, ACM DocEng 2012, SVG Open 2010, EGC 2008, IWH2XP 2004 (Workshop of WWW), ACM DocEng 2004, KDMCD 2002, CIDE 2002, MediaNet 2002, OMTT 2003, MMM 2001, ACM Multimedia 2000, WWW 2000, WWW 1999, ACM Multimedia 1999.

Reviews service : I am regular reviewer for the following journals: TOIS (ACM Transactions on Information Systems), TOIT (ACM Transactions on Internet Technology), TKDE (IEEE Transactions on Knowledge and Data Engineering), IEEE Pervasive Computing, IPL (Information Processing Letters), IEEE Multimedia Magazine, IEEE Multimedia Journal, ACM Multimedia Systems Journal, Computer Journal of he British Computer Society.

3.5 Technology transfer, standards and software

Technology transfer : Science and technology advisor of Raise Partner S.A., a start-up company created at Inria Grenoble Rhône-Alpes research center. I was co-founder of the company and contributed to the design of a high performance web infrastructure for risk management. I worked one day a week for five years (under 25.2 valorization status), from 2005 to 2010.

W3C Standards : I have been a member of the W3C SYMM (Synchronized Multimedia Working Group) from 1996 until 2008. I contributed to the initial founding work of the group, to requirement specifications and was both an author and editor of SMIL 1.0, 2.0 et 2.1 W3C recommendations. I have also contributed to other groups such Device Independence Authoring (DIA) and Composite Capabilities/Preference Profiles (CC/PP).

Software development

XML reasoning solver: A static analyzer for a finite tree logic adapted to XML/XPath. This system allows solving problems such as XPath equivalence, containment and overlap under type constraints (DTDs, XML Schemas, Relax NG). The analyzer is built on top of a finite tree logic solver for a new modal logic equipped with recursion and backward axes. The solver is very fast in practice and uses symbolic techniques (Binary Decision Diagrams). The solver has been recently extended to support functions, parametric function and polymorphic subtyping. The software is developed in java in collaboration with Pierre Genevès and Nils Gesbert.

NAC infrastructure: NAC (Negotiation and Adaptation Core) is a core infrastructure for adaptation and negotiation of multimedia services for heterogeneous environments. The objective of the implemented core is to allow clients (PDA, WAP phones, laptops, etc.) to use multimedia content, which is adapted automatically to their preferences and capacities. Client descriptions (i.e. profiles) are declared in CC/PP structures stored in an XML format and can be modified at anytime. NAC includes two kinds of adaptations: structural adaptation such as adapting XHTML to WML, SMIL and HTML filtering, and media adaptation such as image compression, text to SMS, remote text to speech. The proposed core doesn't make any assumption on the existing platform and browsers. Players must only point the network connection to the server or the proxy that uses the Adaptation and Negotiation Module (ANM). Services are then adapted automatically. The software was developed in java in collaboration with Tayeb Lemlouma.

MIP Phone/PocketSMIL: Is an embedded software for IP telephony based on the Session Initiation Protocol (SIP) for signaling, on real-time streaming protocol (RTSP) for session description and on the real-time transport protocol (RTP) for synchronized audio and video payloads access. This software has been also extended to demonstrate multimedia-messaging (MMS) capabilities using SMIL. The software was written in C++ on Windows CE and was transferred to Cegetel-Innovatel Company, SFR's research and development division. The software was developed in java in collaboration with Daniel Hagimont.

IncXSLT engine: IncXSLT is an incremental transformation processor for XSLT. This software was developed within the Xerces engine of the Apache foundation. A series of authoring tools for several domain-specific DTDs such as ATA and DocBook were developed with incXSLT as a kernel component. The software was developed in java in collaboration with Lionel Villard.

Madeus 1&2: Madeus is an authoring tool integrating authoring and presentation of multimedia documents. It provides document authors with an efficient and flexible way to specify a multimedia document while retaining established declarative languages for temporal synchronization and spatial positioning. The design principles of Madeus rely on constraints. They are used as a document specification language and provide the required authoring abstractions. First, at the interface level, constraints are used to combine intuitive graphical representations of temporal scenarios, like timelines, with the ease of modification by automatic updates through constraint propagation. The tool maintains document consistency in order to prevent authors from introducing errors when writing complex presentations. Constraints are based on a powerful representation of scenarios as graphs, which are also used for scheduling and time-based navigation. The software was developed first in C, then ported to

java. The tool development was in collaboration with Loay Sabry-Ismaïl for version 1, and Lionel Villard, Laurent Tardif and Tien Tran-Thuong, for version 2.

Thot: Thot is a generic system for the development of document-centered applications based on the concept of structured active documents. It can be used for building interactive system as well as automatic processors. It's the result of many years of research at Inria, CNRS and the University of Grenoble (UJF), with a number of academic and industrial collaborations. The Grif commercial products, for instance, are a result of these collaborations. The Amaya tool is also built on top the Thot library for web content editing. My role was the design of a multimedia manipulation library within Thot for supporting multimedia content in structured and web documents.

3.6 Teaching

During my research career, I have regularly taught courses at different levels from undergraduate to graduate in different Universities and Engineering schools. In additional to traditional computer science courses such as operating systems, networking and databases, I have introduced and put in place and enhanced over time new teachings closer to my research activities. In particular, I have introduced master courses on multimedia programming and XML, both from a programming language and database perspectives. My activities covered the entire teaching spectrum from elaborating the courses content, to practical lab manipulations, and exams. These courses have been taught at different locations such as ENST Bretagne, Chambery, Ottawa or Lausanne.

Course	**Level**	**Volume**
Network protocols, routing and flow control.	ENSERG-ENSIMAG	137 h
Web Standards: XML, XSL and SMIL	ENST Bretagne (3ème année), Université d'Ottawa, Canada	52 h
Algorithms, data structures and functional programming	Deug A1, A2, Miass (Université de Savoie)	75 h
Integrated applications and databases	Deug SHS 2ème Année (Université Pierre Mendès-France)	60 h
Operating systems principles and architectures	ENSIMAG, ESIGEC, Maîtrise (Université Joseph Fourier)	60 h
Hardware architecture of computer systems	Licence (Université de Savoie)	10 h
Electronic multimedia documents, SGML and structured authoring	DESS IDC Université Pierre Mendès-France, EPFL (Lausanne)	30 h
Semi-structured data and systems	ENSIMAG (3ème année)	54
Structured multimedia systems	DEA ISC, MW UJF-INPG	30 h
Multimedia systems principles on the web	Master M2R SIGAL Grenoble	18 h

3.7 Author bibliography

Ayars, J., Bulterman, D., Cohen, A., Day, K., Hodge, E., Hoschka, P., Hyche, E., Jourdan, M., Kim, M., Kubota, K., Lanphier, R., Layaïda, N., and al (2001). Synchronized multimedia integration language (SMIL 2.0). W3C recommendation, World Wide Web Consortium. 13

Bárcenas, E., Genevès, P., and Layaïda, N. (2009a). Counting in trees along multidirectional regular paths. In PLAN-X 2009: Proceedings of the ACM SIGPLAN Workshop on Programming Language Techniques for XML, pages 1–10. ACM.

Bárcenas, E., Genevès, P., and Layaïda, N. (2009b). On the analysis of queries with counting constraints. In DocEng 2009: Proceedings of the 2009 ACM Symposium on Document Engineering, pages 21–24. ACM.

Bárcenas, E., Genevès, P., Layaïda, N., and Schmitt, A. (2010). On the count of trees. Research Report 7251, INRIA.

Bárcenas, E., Genevès, P., Layaïda, N., and Schmitt, A. (2011). Query reasoning on trees with types, interleaving, and counting. In IJCAI 2011: Proceedings of the 22nd International Joint Conference on Artificial Intelligence, pages 718–723. IJCAI/AAAI. 21

Bulterman, D., Grassel, G., Jansen, J., Koivisto, A., Layaïda, N., Michel, T., Mullender, S., and Zucker, D. (2005). Synchronized multimedia integration language (SMIL 2.1). W3C recommendation, World Wide Web Consortium. 13

Chekol, M. W., Euzenat, J., Genevès, P., and Layaïda, N. (2011). PSPARQL query containment. In DBPL 2011: Proceedings of the 13th International Symposium on Database Programming Languages.

Deltour, R., Layaïda, N., and Weck, D. (2005). LimSee2: A cross-platform SMIL authoring tool. ERCIM News, (62).

Euzenat, J., Layaïda, N., and Dias, V. (2003). A semantic framework for multimedia document adaptation. In IJCAI 2003: Proceedings of the 18th International Joint Conference on Artificial Intelligence, pages 31–36. Morgan Kauffman. 18

Fargier, H., Jourdan, M., Layaïda, N., and Vidal, T. (1998). Using temporal constraints networks to manage temporal scenario of multimedia documents. In Proceedings of the ECAI-98 International Workshop on Spatial and Temporal Reasoning, pages 1–6. 14, 15

Genevès, P. and Layaïda, N. (2006a). Comparing XML path expressions. In DocEng 2006: Proceedings of the 2006 ACM Symposium on Document Engineering, pages 65–74. ACM.

Genevès, P. and Layaïda, N. (2006b). A decision procedure for XPath containment. Research Report 5867, INRIA.

Genevès, P. and Layaïda, N. (2006c). Mu-calculus based resolution of XPath decision problems. Research Report 5868, INRIA.

Genevès, P. and Layaïda, N. (2006d). A system for the static analysis of XPath. ACM Transactions on Information Systems, 24(4):475–502. 21

Genevès, P. and Layaïda, N. (2007). Deciding XPath containment with MSO. Data and Knowledge Engineering, 63(1):108–136. 21

Genevès, P. and Layaïda, N. (2008a). Static analysis of XML programs. ERCIM News : The Future Web, 2008(72).

Genevès, P. and Layaïda, N. (2008b). XML reasoning solver user manual. Research Report 6726, INRIA.

Genevès, P. and Layaïda, N. (2010a). Eliminating dead-code from xquery programs. In ICSE 2010: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2, pages 305–306. ACM.

Genevès, P. and Layaïda, N. (2010b). XML reasoning made practical. In ICDE 2010: Proceedings of the 26th International Conference on Data Engineering, pages 1169–1172. IEEE.

Genevès, P. and Layaïda, N. (2011). Inconsistent path detection for XML IDEs. In ICSE 2011: Proceedings of the 33rd International Conference on Software Engineering, pages 983–985. ACM. 22

Genevès, P., Layaïda, N., and Quint, V. (2008a). Ensuring query compatibility with evolving XML schemas. Research Report 6711, INRIA.

Genevès, P., Layaïda, N., and Quint, V. (2009). Identifying query incompatibilities with evolving XML schemas. In ICFP 2009: Proceeding of the 14th ACM SIGPLAN International Conference on Functional Programming, pages 221–230. ACM. 22

Genevès, P., Layaïda, N., and Quint, V. (2011). Impact of XML schema evolution. ACM Transactions on Internet Technologies, 11(1):4–27. 22

Genevès, P., Layaïda, N., and Quint, V. (2012). On the analysis of cascading style sheets (to appear). In WWW 2012: Proceedings of the 21st International World Wide Web Conference. ACM. 21

Genevès, P., Layaïda, N., and Schmitt, A. (2007a). Efficient static analysis of XML paths and types. In PLDI 2007: Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation, pages 342–351. ACM. 21

Genevès, P., Layaïda, N., and Schmitt, A. (2007b). XPath typing using a modal logic with converse for finite trees. In PLAN-X 2007: Programming Language Technologies for XML, An ACM SIGPLAN Workshop co-located with POPL 2007, pages 61–72. ACM.

Genevès, P., Layaïda, N., and Schmitt, A. (2008b). Efficient static analysis of XML paths and types. Research Report 6590, INRIA.

Gesbert, N., Genevès, P., and Layaïda, N. (2011). Parametric polymorphism and semantic subtyping: the logical connection. In ICFP 2011: Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming, pages 107–116. ACM. 23

Hagimont, D. and Layaïda, N. (2002). Adaptation de serveur multimédia par un code mobile. Technique et Science Informatiques (TSI), numéro spécial Agents et code mobile, 21(6):877–898. 14, 16

Hoschka, P., Bugaj, S., Bulterman, D., Layaïda, N., and al (1998). Synchronized multimedia integration language (SMIL) 1.0 specification. W3C recommendation, World Wide Web Consortium. 13

Jourdan, M., Layaïda, N., and Roisin, C. (1998a). Handbook of Internet and Multimedia Systems and Applications, part 1: Tools and Standards, chapter A survey on authoring techniques for temporal scenarios of multimedia documents, pages 469–490. CRC Press.

Jourdan, M., Layaïda, N., Roisin, C., Sabry-Ismaïl, L., and Tardif, L. (1998b). Madeus, and authoring environment for interactive multimedia documents. In ACM Multimedia 1998: Proceedings of the Sixth ACM International Conference on Multimedia, pages 267–272. ACM. 12

Jourdan, M., Layaïda, N., and Sabry-Ismaïl, L. (1996). Presentation services in madeus : an authoring environment for multimedia documents. Research Report 2983, INRIA.

Jourdan, M., Layaïda, N., and Sabry-Ismaïl, L. (1997a). MADEUS: An authoring environment for interactive multimedia documents. In ICCC'97: Proceedings of the Thirteenth International Conference on Computer Communications, pages 19–26. IFIP.

Jourdan, M., Layaïda, N., and Sabry-Ismaïl, L. (1997b). MADEUS: An authoring environment for interactive multimedia documents. In ICMCS'97: Proceedings of the IEEE International Conference on Multimedia Computing and Systems, pages 644–645. IEEE.

Jourdan, M., Layaïda, N., and Sabry-Ismaïl, L. (1997c). Time representation and management in madeus: An authoring environment for multimedia documents. In ICMCN 1997: International Conference Multimedia Computing and Networking, volume 3020, pages 68–79. SPIE.

Jourdan, M., Layaïda, N., Sabry-Ismaïl, L., and Roisin, C. (1997d). An integrated authoring and presentation environment for interactive multimedia documents. In MMM'97: Proceedings of the 4th International Conference on Multimedia Modelling, pages 248–263. World Scientific.

Laborie, S., Euzenat, J., and Layaïda, N. (2005). Adapter temporellement un document SMIL. In AFIA 2005: Atelier Connaissance et Documents Temporels, pages 47–58.

Laborie, S., Euzenat, J., and Layaïda, N. (2006a). Adaptation spatiale efficace de documents SMIL. In RFIA 2006: 15e congrès francophone AFRIF-AFIA Reconnaissance des Formes et Intelligence Artificielle, pages 127–136.

Laborie, S., Euzenat, J., and Layaïda, N. (2006b). Adapting the hypermedia structure in a generic multimedia adaptation framework. In SMAP 2006: Proceedings of the First International Workshop on Semantic Media Adaptation and Personalization, pages 62–67. IEEE.

Laborie, S., Euzenat, J., and Layaïda, N. (2006c). A spatial algebra for multimedia document adaptation. In SAMT 2006: Proceedings of the First International Conference on Semantic and Digital Media Technologies - Posters, volume 233, pages 7–8. CEUR-WS.

Laborie, S., Euzenat, J., and Layaïda, N. (2007). Multimedia document summarization based on a semantic adaptation framework. In SADPI 2007: Proceedings of the International Workshop On Semantically Aware Document Processing And Indexing, pages 87–94. ACM.

Laborie, S., Euzenat, J., and Layaïda, N. (2008). Adaptation spatio-temporelle et hypermédia de documents multimédia. In RTE 2008: Atelier Représentation et Raisonnement sur le Temps et l'Espace, pages 1–13. Hermès.

Laborie, S., Euzenat, J., and Layaïda, N. (2009). Semantic multimedia document adaptation with functional annotations. In SMAP 2009: Proceedings of the 4th International Workshop on Semantic Media Adaptation and Personalization, pages 44–49. IEEE.

Laborie, S., Euzenat, J., and Layaïda, N. (2011). Semantic adaptation of multimedia documents. Multimedia Tools and Applications, 55(3):379–398. 18

Layaïda, N. (1996). Issues on temporal representation of multimedia documents. In RTMW'96: International Workshop on Real-Time Multimedia and The World-Wide-Web.

Layaïda, N. (2005a). SMIL 2.1 basic profile and scalability framework. W3C recommendation, World Wide Web Consortium.

Layaïda, N. (2005b). SMIL 2.1 language profile. W3C recommendation, World Wide Web Consortium. 13

Layaïda, N. and Genevès, P. (2010). Debugging standard document formats. In WWW 2010: Proceedings of the 19th International Conference on World Wide Web, pages 1269–1272. ACM. 22

Layaïda, N., Jourdan, M., and Roisin, C. (1999). Le temps dans les documents, volume H 7 228. Techniques de l'Ingénieur, 249 rue de Crimée, 75925 Paris Cedex 19.

Layaïda, N. and Karmouch, A. (1998). SMIL: The world wide web standard of the W3C. In CCBR'98: Proceedings of the 2nd Canadian Conference on Broadband Research. IEEE. 13

Layaïda, N. and Kéramane, C. (1995). Maintaining temporal consistency of multimedia documents. In Effective abstractions in multimedia layout presentation and interaction, Workshop of the ACM Multimedia'95. ACM.

Layaïda, N. and Lemlouma, T. (2003). NAC: An architecture for multimedia content adaptation for mobile devices. ERCIM News, (54).

Layaïda, N., Lemlouma, T., and Quint, V. (2004). Adaptation et multimédia mobile sur le web. In Mcube 2004: Conférence nationale sur le multimédia mobile, pages 34–41.

Layaïda, N., Lemlouma, T., and Quint, V. (2005). NAC, une architecture pour l'adaptation multimédia sur le web. Technique et Science Informatiques, 24(7):789–813. 17

Layaïda, N. and Ossenbruggen, J. V. (2001). SMIL 2.0 language profile. W3C recommendation, World Wide Web Consortium. 2, 13

Layaïda, N. and Roisin, C. (2004). Le temps dans les documents - langage SMIL. In Document numériques – Gestion de contenu, 249 rue de Crimée, 75925 Paris Cedex 19. Techniques de l'Ingénieur.

Layaïda, N., Roisin, C., and Sabry-Ismaïl, L. (2001). Systèmes Multimédias Communicants, chapter Chapitre 6 : Support d'exécution de documents multimédia, pages 197–222. Hermès, Paris, France. 14

Layaïda, N. and Sabry-Ismaïl, L. (1996a). MADEUS : Un modèle de documents multimédia structurés. Technique et Science Informatiques (TSI): numéro thématique Multimédia Collecticiels, 15(9):1227–1257.

Layaïda, N. and Sabry-Ismaïl, L. (1996b). Maintaining temporal consistency of multimedia documents using constraint networks. In ICMCN 1996: International Conference Multimedia Computing and Networking, volume 2667, pages 124–135. SPIE.

Layaïda, N., Sabry-Ismaïl, L., and Roisin, C. (2002). Dealing with uncertain durations in synchronized multimedia presentations. Multimedia Tools and Applications, 18(3):213–231. 14

Layaïda, N. and Vion-Dury, J.-Y. (1994). Interface d'édition de documents structurés multimédia. In IHM'94 : Actes des Sixièmes Journées de l'Ingénierie des Interfaces Homme-Machine, pages 75–80.

Layaïda, N. and Vion-Dury, J.-Y. (1995). Multimedia authoring: A 3d interactive visualization interface based on a structured document model. In HCI'95: Proceedings of the Sixth International Conference on Human-Computer Interaction, pages 313–318. Elsevier Science.

Lemlouma, T. and Layaïda, N. (2001). The negotiation of multimedia content services in heterogeneous environments. In MMM 2001: Proceedings of the 8th International Conference on Multimedia Modeling, pages 187–206. CWI. 17

Lemlouma, T. and Layaïda, N. (2002). Universal profiling for content negotiation and adaptation in heterogeneous environments. In DIWS 2002: W3C Workshop on Delivery Contex.

Lemlouma, T. and Layaïda, N. (2003a). Adapted content delivery for different contexts. In SAINT 2003: Proceedings of the International Symposium on Applications and the Internet, pages 190–197. IEEE. 17

Lemlouma, T. and Layaïda, N. (2003b). Encoding multimedia presentation for user preferences and limited environments. In ICME 2003: Proceedings of IEEE International Conference on Multimedia and Expo, pages 165–168. IEEE Computer Society.

Lemlouma, T. and Layaïda, N. (2003c). Media resources adaptation for limited devices. In ELPUB 2003: Proceedings of the 7th ICCC/IFIP International Conference on Electronic Publishing, pages 209–218. 17

Lemlouma, T. and Layaïda, N. (2003d). SMIL content adaptation for embedded devices. In Proceedings of the European Conference on Synchronized Multimedia Integration Language.

Lemlouma, T. and Layaïda, N. (2004). Context-aware adaptation for mobile devices. In MDM 2004: Proceedings of the 5th IEEE International Conference on Mobile Data Management, pages 106–111. IEEE. 17

Lemlouma, T. and Layaïda, N. (2005). Content interaction and formatting for mobile devices. In DocEng 2005: Proceedings of the 2005 ACM Symposium on Document Engineering, pages 98–100. ACM.

Sabry-Ismaïl, L., Layaïda, N., and Roisin, C. (1997). Navigation in structured multimedia documents using presentation context. In H2PTM'97: Conférence Internationale sur les Hypertextes et Hypermédias, pages 27–38. Hermès.

Schmitz, P., Yu, J., and Layaïda, N. (1999). Synchronized multimedia integration language (SMIL) document object model. W3C note, World Wide Web Consortium. 13

Villard, L. and Layaïda, N. (2002). An incremental XSLT transformation processor for XML document manipulation. In WWW 2002: Proceedings of the 11th International Conference on World Wide Web, pages 474–485. ACM. 19

Villard, L., Roisin, C., and Layaïda, N. (2000). An XML-based multimedia document processing model for content adaptation. In DDEP 2000: Proceedings of 8th International Conference on Digital Documents and Electronic Publishing, volume 2023 of Lecture Notes in Computer Science, pages 104–119. Springer. 12, 13, 19

Vion-Dury, J.-Y. and Layaïda, N. (2003). Containment of XPath expressions: an inference and rewriting based approach. In Proceedings of the Extreme Markup Languages Conference. IDEAlliance. 21

Four

A Logic for Finite Trees

Abstract

We present a sound and complete satisfiability-testing algorithm and its effective implementation for an alternation-free modal μ-calculus with converse, where formulas are cycle-free, and which is interpreted over finite ordered trees. The time complexity of the satisfiability-testing algorithm is $2^{\mathcal{O}(n)}$ in terms of formula size n. The algorithm is implemented using symbolic techniques (BDD). We present crucial implementation techniques and heuristics that we used to make the algorithm as fast as possible in practice. Our implementation is available online, and can be used to solve logical formulas of practically significant size.

4.1 Introduction

This chapter introduces a logic for reasoning over finite trees, a sound and complete decision procedure for checking the satisfiability of a formula of the logic as well as its effective implementation. The logic is a variant of μ-calculus adapted for finite trees and equipped with backward modalities and nominals. Specifically, the logic is an alternation-free modal μ-calculus with converse, where formulas are cycle-free, and which is interpreted over finite ordered trees. The time complexity of the satisfiability-testing algorithm is optimal: $2^{\mathcal{O}(n)}$ in terms of formula size n. We present crucial implementation techniques like the use of symbolic techniques (BDD) and heuristics that we used to make the algorithm as fast as possible in practice. Our implementation is available online, and can be used to solve logical formulas of practically significant size.

4.1.1 Related Work and Motivations

The propositional μ-calculus was introduced as a logic for describing properties of graphs with labeled edges. It was invented by Dana Scott and Jaco de Bakker, and further developed by Dexter Kozen into the version mostly used nowadays [Kozen, 1983]. Several modal logics can be encoded in the μ-calculus, including linear temporal logic, computational tree logic [Clarke & Emerson, 1981], CTL*, and propositional dynamic logic [Fischer & Ladner, 1979]. In contrast with the importance and large applicative spectrum of the μ-calculus satisfiability problem, only very few actual effective implementations have been reported in the literature. The work found in [Tanabe et al., 2005] points this out neatly: "the satisfiability testing problem for the μ-calculus is known

to be decidable for a variety of extensions and subfragments, but effective implementation has not necessarily been developed for all such logics".

We review below the works that are most closely related in terms of supported logical features (e.g., backward modalities, nominals), models of the logic (trees), or from the point-of-view of the approach oriented toward an effective implementation (effective algorithmics). For instance, the work found in [Pan et al., 2006] pursues a goal similar to ours for the modal logic K. The approach yields effective BDD-based decision procedures for K, usable in practice. However, the expressive power of the logic is incomparable to the one of the μ-calculus since K lacks recursion (no fixpoint) and backward modalities.

Backward Modalities

In applications, we often need to follow edges not only in the forward direction but also in the backward direction. Therefore a research effort has been focusing on temporal logics that can handle both directions of edges as modalities in order to reason about both the "past" and the "future". Although converse modalities do not provide additional expressive power, they provide an advance in terms of succinctness as they offer a notation for otherwise exponentially larger formulas. Succinctness is a crucial matter when considering combined complexity of the decision procedure. The satisfiability problem for the general μ-calculus with converse modalities (MC) is known to be EXPTIME-complete [Vardi, 1998]. The decision procedure is constructed by converting the problem into the emptiness problem of the language recognized by a certain alternating tree automaton on infinite trees. In order to solve the emptiness problem, complex operations are required including determinization of parity automata [Safra, 1988]. No implementation is reported.

The best known complexity for deciding MC is obtained through reduction to the emptiness problem of alternating tree automata on infinite trees, which can be done in $2^{\mathcal{O}(n^4 \cdot \log\, n)}$, where n is the size of the formula [Grädel et al., 2002]. Again, no actual implementation has been reported.

A notable exception is the work found in [Tanabe et al., 2005, 2008], that provides an implementation of a decision procedure for the alternation-free fragment of the μ-calculus with converse (AFMC), whose time complexity is $2^{\mathcal{O}(n \cdot \log\, n)}$, noticing in passing that the decision procedure for the AFMC is less complex than the one for the MC, as expected. The alternation-free restriction makes much sense since the expressive power of AFMC exactly corresponds to the one of weak monadic-second order logic [Kupferman & Vardi, 1999].

Trees

In applications of the satisfiability-checking problem, relevant models often consist only of the set of finite trees (see, e.g., [Zee et al., 2008]). Therefore, even if the AFMC lacks the finite model property (which is lost due to the addition of converse modalities), it makes sense to search for finite trees satisfying a given logical formula.

In this line of research, the work of [Afanasiev et al., 2005] presents a special version of PDL for reasoning about finite sibling-ordered trees. However, the

precise expressive power of the logic is still an open problem, although the logic is subsumed by the AFMC.

In [Tanabe et al., 2005, 2008], models of the logic are Kripke structures (infinite graphs). Owing to an additional logical formula that encodes König's lemma, models can be restricted to be binary-branching finite trees. However, the authors notice that the performance of the decision procedure may not be very attractive in this setting [Tanabe et al., 2005]. The authors do not comment on the reasons, but our research gave us insights. Specifically, a first source of inefficiency of this approach comes from the fact that the decision procedure requires expensive cycle-detection for rejecting infinite derivation paths for least fixpoint formulas. A second and even more fundamental source of inefficiency is that the decision procedure of [Tanabe et al., 2005] must compute a greatest fixpoint: it starts from all possible (graph) nodes and progressively removes all inconsistent nodes until a fixpoint is reached. Finally, if the fixpoint contains a satisfying (tree) structure then the formula is judged as satisfiable. As a consequence, and unlike the algorithm presented in this article, (1) the algorithm must always explore all nodes, and (2) it cannot terminate until full completion of the fixpoint computation (otherwise inconsistencies may remain). The present work shows how this can be avoided for finite trees. As a consequence, the resulting performance of the decision procedure proposed in this article, whose time complexity is $2^{\mathcal{O}(n)}$, is much more attractive.

In an earlier work, a logic for finite trees was presented [Tozawa, 2004], but the logic is not closed under negation.

The connection with Automata

In our extended abstract [Genevès et al., 2007b], we showed the decidability in time $2^{\mathcal{O}(n)}$ of the cycle-free fragment of the AFMC for finite trees. Since then, alternative and closely related approaches based on tree automata have been proposed with similar or higher complexity, but without implementation [Calvanese et al., 2008; Libkin & Sirangelo, 2008; Calvanese et al., 2009; Libkin & Sirangelo, 2010; Calvanese et al., 2010].

Automata-based approaches based on alternating two-way tree automata (2ATA) for infinite trees have resisted implementation, as noticed in [Calvanese et al., 2009], mainly because of complex determinization and parity games (see [Calvanese et al., 2008, 2009], in which it is also mentioned that it is practically infeasible to apply the symbolic approach in the the infinite tree setting).

A more appropriate automata version for finite trees is called weak alternating two-way tree automata (2WATA) which are simpler when compared to infinite tree automata-theoretic techniques. However, they require a conversion to non-deterministic finite tree automata (NTFA) for testing non-emptiness. The translation given in [Calvanese et al., 2010] yields an automaton with $2^{\mathcal{O}(n^2)}$ states in terms of the number n of states of the original 2WATA.

In fact, neither of the papers [Libkin & Sirangelo, 2008; Calvanese et al., 2008, 2009; Libkin & Sirangelo, 2010; Calvanese et al., 2010] provide an implementation. [Calvanese et al., 2008] even remarks that a naive implementation

of their technique would result in a blow-up in complexity, requiring the use of more elaborate techniques very similar to what we have done.

In [Libkin & Sirangelo, 2010], the authors acknowledge the fact that they provide an alternative version of our pioneering work described in our extended abstract [Genevès et al., 2007b]. 2WATA are interesting to shorten some proofs but they do not simplify the implementation.

The present work can be regarded as the pioneering and only efficient implementation of the logic or, alternatively, of the 2WATA framework.

For the sake of simplicity and uniformity between the satisfiability algorithm, the proofs, and the implementation techniques, in the whole present chapter we focus on the native modal logic in the finite case. This also emphasizes the fact that bottom-up construction of the finite tree model and cycle-freeness come naturally and show exactly why the whole approach is efficient.

4.1.2 Contributions

Our main result is a satisfiability-testing algorithm for a logic for finite trees whose time complexity is optimal: $2^{\mathcal{O}(n)}$ in terms of the formula size n, together with its effective implementation through BDD techniques.

The essence of our results lives in a sub-logic of the AFMC, with a syntactic restrictions called cycle-freeness on formulas, and whose models are finite trees. Such restrictions are interesting from a theoretical point of view: we prove that, under these conditions, the least and greatest fixpoint operators collapse in a single fixpoint operator. This makes our logic closed under negation, and also provide many opportunities to derive an efficient implementation.

The decision procedure is implemented and an online demonstration is publicly available, as detailed in §4.5.5.

An extended abstract of this work was presented at the ACM Conference on Programming Language Design and Implementation (PLDI), 2007 [Genevès et al., 2007b]. The new material included in this chapter comprises the following. The notion of cycle-freeness, a fundamental aspect of our logic, and its formalization are much more detailed. Proofs have been added. A detailed run of the algorithm is described. Implementation techniques and the optimizations used to obtain a satisfiability-testing algorithm that performs well in practice are also discussed in more details. More applications have also been added.

4.1.3 Outline

The chapter is organized as follows. We first present our data model, trees with focus, in §4.2. We then introduce the logic in §4.3. Our satisfiability algorithm is introduced and proven correct in §4.4. Crucial implementation techniques are discussed in §4.5. Applications such as Regular Language Equivalence and XPath typing are reviewed in §4.6 together with augmented XQuery IDEs 4.7 and dead-code analysis for XQuery 4.8. We conclude in §4.9.

4.2 Trees with Focus

In order to represent trees that are easy to navigate, we use focused trees, inspired by Huet's Zipper data structure [Huet, 1997]. Focused trees not only describe a tree but also its context: its previous siblings and its parent, including its parent context recursively. Exploring such a structure has the advantage to preserve all information, which is quite useful when considering forward and backward navigation.

Formally, we assume an alphabet Σ of labels, ranged over by σ. The syntax of our data model is as follows.

$$
\begin{array}{rcll}
t & ::= & \sigma[tl] & \text{tree} \\
tl & ::= & & \text{list of trees} \\
 & & \epsilon & \quad \text{empty list} \\
 & | & t :: tl & \quad \text{cons cell} \\
c & ::= & & \text{context} \\
 & & (tl, \mathit{Top}, tl) & \quad \text{root of the tree} \\
 & | & (tl, c[\sigma], tl) & \quad \text{context node} \\
f & ::= & (t, c) & \text{focused tree}
\end{array}
$$

A focused tree (t, c) is a pair consisting of a tree t and its context c. The context $(tl, c[\sigma], tl)$ comprises three components: a list of trees at the left of the current tree in reverse order (the first element of the list is the tree immediately to the left of the current tree), the context above the tree, and a list of trees at the right of the current tree. The context above the tree may be *Top* if the current tree is at the root, otherwise it is of the form $c[\sigma]$ where σ is the label of the enclosing element and c is the context in which the enclosing element occurs.

In order to deal with decision problems such as containment of queries (i.e. binary relations over tree nodes), we need to represent in a focused tree the place where the evaluation of a query was started. To this end, we use a start mark, often simply called "mark" in the following. We thus consider focused trees where a single tree or a single context node is marked, as in $\sigma^{\textcircled{S}}[tl]$ or $(tl, c[\sigma^{\textcircled{S}}], tl)$. When the presence of the mark is unknown, we write it as $\sigma^{\circ}[tl]$. We write $\mathcal{F}$ for the set of finite focused trees containing a single mark. The name of a focused tree is defined as $\mathtt{nm}(\sigma^{\circ}[tl], c) = \sigma$.

We now describe how to navigate focused trees, in binary style. There are four directions, or modalities, that can be followed: for a focused tree f, $f\,\langle 1\rangle$ changes the focus to the first child of the current tree, $f\,\langle 2\rangle$ changes the focus to the next sibling of the current tree, $f\,\langle \overline{1}\rangle$ changes the focus to the parent of the tree if the current tree is a leftmost sibling, and $f\,\langle \overline{2}\rangle$ changes the focus to the previous sibling.

$\mathcal{L}_\mu \ni \varphi, \psi$	$::=$		formula
		$\top$	true
	$\mid$	$p \mid \neg p$	atomic proposition (negated)
	$\mid$	$Ⓢ \mid \neg Ⓢ$	start proposition (negated)
	$\mid$	X	variable
	$\mid$	$\varphi \vee \psi$	disjunction
	$\mid$	$\varphi \wedge \psi$	conjunction
	$\mid$	$\langle a \rangle \varphi \mid \neg \langle a \rangle \top$	existential (negated)
	$\mid$	$\mu \overline{X_i.\varphi_i}$ in ψ	least n-ary fixpoint
	$\mid$	$\nu \overline{X_i.\varphi_i}$ in ψ	greatest n-ary fixpoint

Figure 4.1: Logic formulas

Formally, we have:

$$(\sigma^\circ[t :: tl], c)\langle 1 \rangle \stackrel{\text{def}}{=} (t, (\epsilon, c[\sigma^\circ], tl))$$
$$(t, (tl_l, c[\sigma^\circ], t' :: tl_r))\langle 2 \rangle \stackrel{\text{def}}{=} (t', (t :: tl_l, c[\sigma^\circ], tl_r))$$
$$(t, (\epsilon, c[\sigma^\circ], tl))\langle \overline{1} \rangle \stackrel{\text{def}}{=} (\sigma^\circ[t :: tl], c)$$
$$(t', (t :: tl_l, c[\sigma^\circ], tl_r))\langle \overline{2} \rangle \stackrel{\text{def}}{=} (t, (tl_l, c[\sigma^\circ], t' :: tl_r))$$

When the focused tree does not have the required shape, these operations are not defined.

4.3 The Logic

We introduce the logic as a sub-logic of the alternation free modal μ-calculus with converse. We also introduce a restriction on the formulas we consider and give an interpretation of formulas as sets of finite focused trees. We finally show that this restriction and this interpretation make the greatest and smallest fixpoint collapse, yielding a logic that is closed under negation.

4.3.1 Formulas

In the following, we use an overline bar to denote tuples. For instance, we write $\overline{X_i = \varphi_i}$ for $(X_1 = \varphi_1; X_2 = \varphi_2; \ldots; X_n = \varphi_n)$. Tuples of variables, such as $\overline{X_i}$, are often identified to sets.

In the following definitions, $a \in \{1, 2, \overline{1}, \overline{2}\}$ are programs. Atomic propositions p correspond to labels from Σ. We also assume that $\overline{\overline{a}} = a$. Formulas defined in Figure 4.1 include the truth predicate, atomic propositions (denoting the name of the tree in focus), start propositions (denoting the presence of the start mark), disjunction and conjunction of formulas, formulas under an existential (denoting the existence of a subtree satisfying the sub-formula), and least and greatest n-ary fixpoints. We chose to include a n-ary version of fixpoints because regular types are often defined as a set of mutually recursive definitions, making their translation in our logic more direct and succinct. In the following we write "$\mu X.\varphi$" for "$\mu \overline{X.\varphi}$ in φ".

$$
\begin{aligned}
[\![\top]\!]_V &\stackrel{\text{def}}{=} \mathcal{F} & [\![p]\!]_V &\stackrel{\text{def}}{=} \{f \mid \mathtt{nm}(f) = p\} \\
[\![X]\!]_V &\stackrel{\text{def}}{=} V(X) & [\![\neg p]\!]_V &\stackrel{\text{def}}{=} \{f \mid \mathtt{nm}(f) \neq p\} \\
[\![\varphi \vee \psi]\!]_V &\stackrel{\text{def}}{=} [\![\varphi]\!]_V \cup [\![\psi]\!]_V & [\![\textcircled{\scriptsize S}]\!]_V &\stackrel{\text{def}}{=} \left\{f \mid f = (\sigma^{\textcircled{\scriptsize S}}[tl], c)\right\} \\
[\![\varphi \wedge \psi]\!]_V &\stackrel{\text{def}}{=} [\![\varphi]\!]_V \cap [\![\psi]\!]_V & [\![\neg\textcircled{\scriptsize S}]\!]_V &\stackrel{\text{def}}{=} \{f \mid f = (\sigma[tl], c)\}
\end{aligned}
$$

$$
\begin{aligned}
[\![\langle a \rangle\, \varphi]\!]_V &\stackrel{\text{def}}{=} \{f\,\langle \overline{a} \rangle \mid f \in [\![\varphi]\!]_V \wedge f\,\langle \overline{a} \rangle \text{ defined}\} \\
[\![\neg\,\langle a \rangle\, \top]\!]_V &\stackrel{\text{def}}{=} \{f \mid f\,\langle a \rangle \text{ undefined}\} \\
[\![\mu\overline{X_i.\varphi_i} \text{ in } \psi]\!]_V &\stackrel{\text{def}}{=} \text{let } \overline{T_i} = \left(\bigcap \left\{\overline{T_i \subseteq \mathcal{F}} \mid \overline{[\![\varphi_i]\!]_{V[\overline{T_i/X_i}]} \subseteq T_i}\right\}\right)_i \\
&\qquad \text{in } [\![\psi]\!]_{V[\overline{T_i/X_i}]} \\
[\![\nu\overline{X_i.\varphi_i} \text{ in } \psi]\!]_V &\stackrel{\text{def}}{=} \text{let } \overline{T_i} = \left(\bigcup \left\{\overline{T_i \subseteq \mathcal{F}} \mid \overline{T_i \subseteq [\![\varphi_i]\!]_{V[\overline{T_i/X_i}]}}\right\}\right)_i \\
&\qquad \text{in } [\![\psi]\!]_{V[\overline{T_i/X_i}]}
\end{aligned}
$$

Figure 4.2: Interpretation of formulas

4.3.2 Model

We define in Figure 4.2 an interpretation of our formulas as subsets of $\mathcal{F}$, the set of finite focused trees with a single start mark. The interpretation of the n-ary fixpoints first compute the smallest or largest interpretation for each φ_i, bind the resulting sets T_i to the variables X_i, then returns the interpretation of ψ.

To illustrate the interpretation of fixpoints, consider the two following formulas $\varphi = \mu X.\, \langle 1 \rangle\, X \vee \langle \overline{1} \rangle\, X$ and $\psi = \nu X.\, \langle 1 \rangle\, X \vee \langle \overline{1} \rangle\, X$, which respectively expand to $\overline{\mu X.\, \langle 1 \rangle\, X \vee \langle \overline{1} \rangle\, X}$ in $\langle 1 \rangle\, X \vee \langle \overline{1} \rangle\, X$ and $\overline{\nu X.\, \langle 1 \rangle\, X \vee \langle \overline{1} \rangle\, X}$ in $\langle 1 \rangle\, X \vee \langle \overline{1} \rangle\, X$.

The interpretation of φ is straightforward: associating the empty set to X, we have

$$[\![\langle 1 \rangle\, X \vee \langle \overline{1} \rangle\, X]\!]_{[\emptyset/X]} \subseteq \emptyset$$

thus $[\![\varphi]\!] = \emptyset$. Intuitively, there is no base case in the formula, hence the smallest fixpoint is the empty one.

The interpretation of ψ is more complex: it is the set of every focused tree with at least two nodes, one being the parent of the other. We now show that the interpretation of ψ includes the focused tree $f_1 = (a[b[\epsilon]], T)$, where T is the top-level context $(\epsilon, \mathit{Top}, \epsilon)$. We do not specify the position of the mark as it is not used in the query: it could be anywhere. Let $f_2 = f_1\,\langle 1 \rangle$, that is the tree $(b[\epsilon], (\epsilon, T[a], \epsilon))$. We thus have $f_2\,\langle \overline{1} \rangle = f_1$. Finally, let V be the mapping

$[\{f_1; f_2\}/X]$. We compute as follow:

$$\begin{aligned}
&[\![\langle 1\rangle X \vee \langle \overline{1}\rangle X]\!]_V \\
&= [\![\langle 1\rangle X]\!]_V \cup [\![\langle \overline{1}\rangle X]\!]_V \\
&= \{f\langle \overline{1}\rangle \mid f \in [\![X]\!]_V \wedge f\langle \overline{1}\rangle \text{ defined}\} \cup \{f\langle 1\rangle \mid f \in [\![X]\!]_V \wedge f\langle 1\rangle \text{ defined}\} \\
&= \{f_1\} \cup \{f_2\}
\end{aligned}$$

thus $V(X) \subseteq [\![\langle 1\rangle X \vee \langle \overline{1}\rangle X]\!]_V$, hence by definition of the largest fixpoint, we have $f_1 \in [\![\psi]\!]_\emptyset$.

We now state a very simple property of fixpoints: the interpretation of a formula is equal to the interpretation of any of its unfoldings.

Definition 4.3.1 (Unfolding of a formula)**.** *The unfolding of a formula φ is the set $unf(\varphi)$ inductively defined as*

$$\begin{aligned}
unf(\varphi) &\stackrel{def}{=} \{\varphi\} \text{ for } \varphi = \top, p, \neg p, Ⓢ, \neg Ⓢ, X, \neg\langle a\rangle \top \\
unf(\varphi \vee \psi) &\stackrel{def}{=} \{\varphi' \vee \psi' \mid \varphi' \in unf(\varphi), \psi' \in unf(\psi)\} \\
unf(\varphi \wedge \psi) &\stackrel{def}{=} \{\varphi' \wedge \psi' \mid \varphi' \in unf(\varphi), \psi' \in unf(\psi)\} \\
unf(\langle a\rangle \varphi) &\stackrel{def}{=} \{\langle a\rangle \varphi' \mid \varphi' \in unf(\varphi)\} \\
unf(\mu\overline{X_i.\varphi_i} \text{ in } \psi) &\stackrel{def}{=} unf(\psi\{^{\mu\overline{X_i.\varphi_i} \text{ in } \varphi_i}/_{X_i}\}) \cup \{\mu\overline{X_i.\varphi_i} \text{ in } \psi\} \\
unf(\nu\overline{X_i.\varphi_i} \text{ in } \psi) &\stackrel{def}{=} unf(\psi\{^{\nu\overline{X_i.\varphi_i} \text{ in } \varphi_i}/_{X_i}\}) \cup \{\nu\overline{X_i.\varphi_i} \text{ in } \psi\}
\end{aligned}$$

Proposition 4.3.2. *Let φ be a formula, for every unfolding $\psi \in unf(\varphi)$, we have $[\![\varphi]\!]_V = [\![\psi]\!]_V$.*

4.3.3 Cycle-Free Formulas

As shown in §4.3.2, the smallest and greatest fixpoints do not coincide. We now introduce a restriction that will make them collapse, requiring formulas to be cycle-free. To define this notion, we first need to introduce the set of paths of a formula. Given a formula φ, the set of its paths $\mathcal{P}(\varphi)$ is the set of sequential chains of modalities contained in the formula. Writing ϵ for the empty path, we have the following.

$$\begin{aligned}
\mathcal{P}(\langle a\rangle \varphi) &= \{\langle a\rangle p \mid p \in \mathcal{P}(\varphi)\} \\
\mathcal{P}(\varphi \vee \psi) &= \mathcal{P}(\varphi) \cup \mathcal{P}(\psi) \\
\mathcal{P}(\varphi \wedge \psi) &= \mathcal{P}(\varphi) \cup \mathcal{P}(\psi) \\
\mathcal{P}(\varphi) &= \epsilon \qquad \text{otherwise}
\end{aligned}$$

Note that this notion is very syntactic, and unfolding a fixpoint in a formula may change its set of paths.

A modality cycle in a path is a sub-sequence of the form $\langle a\rangle\langle \overline{a}\rangle$. We new define cycle-free formulas as formulas for which there is a bound in the number of modality cycles of their paths, independent on the unfolding.

$$\frac{\varphi = \top, p, \neg p, \textcircled{S}, \text{ or } \neg\textcircled{S}}{\Delta \parallel \Gamma \vdash_I^R \varphi} \qquad \frac{\Delta \parallel \Gamma \vdash_I^R \varphi \qquad \Delta \parallel \Gamma \vdash_I^R \psi}{\Delta \parallel \Gamma \vdash_I^R \varphi \vee \psi}$$

$$\frac{\Delta \parallel \Gamma \vdash_I^R \varphi \qquad \Delta \parallel \Gamma \vdash_I^R \psi}{\Delta \parallel \Gamma \vdash_I^R \varphi \wedge \psi} \qquad \frac{}{\Delta \parallel \Gamma \vdash_I^R \neg \langle a \rangle \top} \qquad \frac{\Delta \parallel (\Gamma \lhd \langle a \rangle) \vdash_I^R \varphi}{\Delta \parallel \Gamma \vdash_I^R \langle a \rangle \varphi}$$

$$\frac{\forall X_j \in \overline{X_i}.\ \left((\Delta + \overline{X_i : \varphi_i}) \parallel (\Gamma + \overline{X_i : \_}) \vdash_{I \setminus \overline{X_i}}^{R \setminus \overline{X_i}} \varphi_j\right) \qquad \Delta \parallel \Gamma \vdash_{I \cup \overline{X_i}}^{R \setminus \overline{X_i}} \psi}{\Delta \parallel \Gamma \vdash_I^R \mu \overline{X_i.\varphi_i} \text{ in } \psi}$$

$$\frac{\forall X_j \in \overline{X_i}.\ \left((\Delta + \overline{X_i : \varphi_i}) \parallel (\Gamma + \overline{X_i : \_}) \vdash_{I \setminus \overline{X_i}}^{R \setminus \overline{X_i}} \varphi_j\right) \qquad \Delta \parallel \Gamma \vdash_{I \cup \overline{X_i}}^{R \setminus \overline{X_i}} \psi}{\Delta \parallel \Gamma \vdash_I^R \nu \overline{X_i.\varphi_i} \text{ in } \psi}$$

$$\textsc{NoRec}\ \frac{X \in R \qquad \Gamma(X) = \langle a \rangle}{\Delta \parallel \Gamma \vdash_I^R X} \qquad \textsc{Rec}\ \frac{X \notin R \qquad \Delta \parallel \Gamma \vdash_I^{R \cup \{X\}} \Delta(X)}{\Delta \parallel \Gamma \vdash_I^R X} \qquad \textsc{Ign}\ \frac{X \in I}{\Delta \parallel \Gamma \vdash_I^R X}$$

Figure 4.3: Deciding Cycle-free Formulas

Definition 4.3.3 (Cycle-free formula)**.** *A formula φ is cycle-free iff there exists an integer n such that for any unfolding $\psi \in unf(\varphi)$, for any path $p \in \mathcal{P}(\psi)$, the number of modality cycles in p is strictly smaller than n.*

For instance, the formula "$\mu \overline{X = \langle 1 \rangle (\top \vee \langle \overline{1} \rangle X)}$ in X" is not cycle free: for any integer n, there is an unfolding of the formula such that a path with n modality cycles exists. Similarly, the formulas φ and ψ in §4.3.2 are also not cycle free. On the other hand, "$\mu \overline{X = \langle 1 \rangle (X \vee Y),\ Y = \langle \overline{1} \rangle (Y \vee \top)}$ in X" formula is cycle free: there is at most one modality cycle for each path, independently of the number of unfoldings of its fixpoint.

Cycle-free formulas have a very interesting property, which we now describe. To test whether a tree satisfies a formula, one may define a straightforward inductive relation between trees and formulas that only holds when the root of the tree satisfies the formula, unfolding fixpoints if necessary. Given a tree, if a formula φ is cycle free, then every node of the tree will be tested a finite number of time against any given subformula of φ. The intuition behind this property, which holds a central role in the proof of lemma 4.3.6, is the following. If a tree node is tested an infinite number of times against a subformula, then there must be a cycle in the navigation in the tree, corresponding to some modalities occurring in the subformula, between one occurrence of the test and the next one. As we consider trees, the cycle implies there is a modality cycle in the formula (as unbalanced cycles of the form $\langle 1 \rangle \langle 2 \rangle \langle \overline{1} \rangle \langle \overline{2} \rangle$ cannot occur). Hence the number of modality cycles in any expansion of φ is unbounded, thus the formula is not cycle free.

Although it provides the correct intuition, Definition 4.3.3 is not very prac-

tical. We give in Figure 4.3 a decision procedure, in the form of an inductive relation, that ensures that a formula is cycle free. In the judgement $\Delta \parallel \Gamma \vdash_I^R \varphi$ of Figure 4.3, Δ is an environment binding some recursion variables to their formulas, Γ binds variables to modalities, R is a set of variables that have already been expanded (see below), and I is a set of variables already checked.

The environment Γ used to derive the judgement consists of bindings from variables (from enclosing fixpoint operators) to modalities. A modality may be $\_$ (no information is known about the variable), $\langle a \rangle$ (the last modality taken $\langle a \rangle$ was consistent), or $\bot$ (a cycle has been detected). A formula is not cycle free if an occurrence of a variable under a fixpoint operator is either not under a modality (in this case $\Gamma(X) = \_$), or is under a cycle ($\Gamma(X) = \bot$). Cycle detection uses an auxiliary operator to detect modality cycles:

$$\Gamma \lhd \langle a \rangle \stackrel{\text{def}}{=} \{X : (\Gamma(X) \lhd \langle a \rangle)\}$$

where

$\cdot \lhd \cdot$	$\langle 1 \rangle$	$\langle 2 \rangle$	$\langle \overline{1} \rangle$	$\langle \overline{2} \rangle$
$\_$	$\langle 1 \rangle$	$\langle 2 \rangle$	$\langle \overline{1} \rangle$	$\langle \overline{2} \rangle$
$\langle 1 \rangle$	$\langle 1 \rangle$	$\langle 2 \rangle$	$\bot$	$\langle \overline{2} \rangle$
$\langle 2 \rangle$	$\langle 1 \rangle$	$\langle 2 \rangle$	$\langle \overline{1} \rangle$	$\bot$
$\langle \overline{1} \rangle$	$\bot$	$\langle 2 \rangle$	$\langle \overline{1} \rangle$	$\langle \overline{2} \rangle$
$\langle \overline{2} \rangle$	$\langle 1 \rangle$	$\bot$	$\langle \overline{1} \rangle$	$\langle \overline{2} \rangle$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

To check that mutually recursive formulas are cycle-free, we proceed the following way. When a mutually recursive formula is encountered, for instance $\mu\overline{X_i.\varphi_i}$ in ψ, we check every recursive binding. Because of mutual recursion, we cannot check formulas independently and we need to expand a variable the first time it is encountered (rule Rec). However there is no need to expand it a second time (rule NoRec). When checking ψ, as the formulas bound to the enclosing recursion have been checked to be cycle free, there is no need to further check these variables (rule Ign). To account for shadowing of variables, we make sure that newly bound recursion variables are removed from I and R when checking a recursion. One may easily prove that if $\Delta \parallel \Gamma \vdash_I^R \varphi$ holds, then $I \cap R = \emptyset$.

This relation detects when a formula is not cycle free because, in this case, there must be a recursive binding of X_i to φ_i such that $\varphi_i\{^{\varphi_i}/_{X_i}\}\{^{\overline{\varphi_j}}/_{\overline{X_j}}\}$ exhibits a modality cycle above X_i, where the X_j are other recursion variables already defined (either in the recursion defining X_i or in an enclosing recursion definition). Cycles are thus detected unfolding every recursive definition once in every formula.

Note that we may wrongly detect a formula as having a cycle. For instance, the formula $\mu X.\, \langle 1 \rangle \langle \overline{1} \rangle X$ in $\top$ is said to include a cycle even though the variable on which the cycle occurs never needs to be expanded. We have found that in practice this approximation is precise enough to check formulas entered by hand. We state that our approximation is correct.

Lemma 4.3.4. *Let φ be a formula. If $\emptyset \parallel \emptyset \vdash^{\emptyset}_{\emptyset} \varphi$, then φ is cycle-free.*

Proof: [Sketch] We proceed by contraposition: we assume φ is not cycle-free, and show that we cannot derive $\emptyset \parallel \emptyset \vdash^{\emptyset}_{\emptyset} \varphi$. As φ is not cycle-free, it is because a modality and its inverse are under a recursion (either directly, or through a conjunction or disjunction), or because a recursion variable is not guarded by a modality. Focusing on this fixpoint, we can show that after expanding the variable (rule Rec), when we encounter the variable again (rule NoRec), we then either have $\Gamma(X) = \bot$ or $\Gamma(X) = \_$, thus the derivation is not possible. □

We are now ready to show a first result: in the finite focused-tree interpretation, the least and greatest fixpoints coincide for cycle-free formulas. To this end, we prove a stronger result that states that a given focused tree is in the interpretation of a cycle-free formula φ if it is in the interpretation of a finite unfolding of the formula $unf_f(\varphi)$. The definition of finite unfolding below is very similar to Definition 4.3.1. The only difference is in the handling of a fixpoint: the fixpoint itself is not included in the set of unfoldings. As a consequence, formulas in $unf_f(\varphi)$ do not contain any fixpoint operator and correspond to the finite unfoldings of φ followed by the erasure of its fixpoints. Note that if there is no base case to a fixpoint of a formula, as in $\mu X.\, \langle 1 \rangle \langle \overline{1} \rangle X$ in X, then the finite unfolding of this formula will be the empty set.

Definition 4.3.5 (Finite unfolding)**.** *The finite unfolding of a formula φ is the smallest set $unf_f(\varphi)$ inductively defined as*

$$
\begin{aligned}
unf_f(\varphi) &\stackrel{def}{=} \{\varphi\} \text{ for } \varphi = \top, p, \neg p, \textcircled{S}, \neg\textcircled{S}, X, \neg\langle a \rangle \top \\
unf_f(\varphi \vee \psi) &\stackrel{def}{=} \{\varphi' \vee \psi' \mid \varphi' \in unf_f(\varphi), \psi' \in unf_f(\psi)\} \\
unf_f(\varphi \wedge \psi) &\stackrel{def}{=} \{\varphi' \wedge \psi' \mid \varphi' \in unf_f(\varphi), \psi' \in unf_f(\psi)\} \\
unf_f(\langle a \rangle \varphi) &\stackrel{def}{=} \{\langle a \rangle \varphi' \mid \varphi' \in unf_f(\varphi)\} \\
unf_f(\mu\overline{X_i.\varphi_i} \text{ in } \psi) &\stackrel{def}{=} unf_f(\psi\{^{\mu\overline{X_i.\varphi_i} \text{ in } \varphi_i}/_{X_i}\}) \\
unf_f(\nu\overline{X_i.\varphi_i} \text{ in } \psi) &\stackrel{def}{=} unf_f(\psi\{^{\nu\overline{X_i.\varphi_i} \text{ in } \varphi_i}/_{X_i}\})
\end{aligned}
$$

Lemma 4.3.6. *Let φ a cycle-free formula, then we have the following.*

$$[\![\varphi]\!]_V = \bigcup_{\psi \in unf_f(\varphi)} [\![\psi]\!]_V$$

The intuition why this lemma holds is the following. Given a tree satisfying φ, we deduce from the hypothesis that φ is cycle free the fact that every node of the tree will be tested a finite number of times against every subformula of φ. As the tree and the number of subformulas are finite, the satisfaction derivation is finite hence only a finite number of unfolding is necessary to prove that the tree satisfies the formula. As least and greatest fixpoints coincide when only a finite number of unfolding is required, this is sufficient to show that they collapse. Note that this would not hold if infinite trees were allowed: the

formula $\mu X.\langle 1\rangle X$ is cycle free, but its interpretation is empty, whereas the interpretation of $\nu X.\langle 1\rangle X$ includes every tree with an infinite branch of $\langle 1\rangle$ children.

of Lemma 4.3.6. Let f in $[\![\varphi]\!]_V$, we show that it is in $[\![\psi]\!]_V$ for some finite unfolding $\psi \in unf_f(\varphi)$. As recursive definitions are never negated, the converse is immediate.

As hinted above, the result is a consequence of the fact that a sub-formula is never confronted twice to the same node of f as there is no cycle in the formula. It is thus possible to annotate occurrences of ν and μ with the direction the formula is exploring for each variable, as in Figure 4.3, and prove the result by induction on the size of f in this direction.

First, we unfold every formula once, to guarantee that the sub-formulas of the shape $\mu\overline{X_i.\varphi_i}$ in ψ are in fact of the shape $\mu\overline{X_i.\varphi_i}$ in φ_j.

Then, we associate each recursion variable in every μ and ν of the initial formula with a unique identifier. (From now on, we do not distinguish between smallest and largest fixed points, as we handle them identically.) For every recursive formula $\mu\overline{X_i.\varphi_i}$ in ψ, we annotate every modality $\langle a\rangle\xi$ in every φ_j where X_i is free in ξ with the variable X_i. Note that modalities may be annotated with more than one variable.

We now detail how recursion identifiers and annotations are updated upon unfolding and encountering modalities.

- Upon unfolding a recursive formula for the first time, the recursion identifiers are recorded and associated with the _ direction. Moreover, they are also associated with an integer, the size of the tree f.
- Upon encountering a modality $\langle a\rangle$ annotated with identifiers, the direction of the identifiers is updated with the modality according to the $\cdot\triangleleft\langle a\rangle$ operator. As the formula is cycle-free, the resulting direction cannot be $\bot$.
- Upon unfolding a recursive formula $\mu\overline{X_i.\varphi_i}$ in φ_j whose identifiers have been already recorded, the integer associated to X_j is updated to be the longest path, defined below, of the current focused tree in X_j's direction. As the formula is cycle-free, a direction must have been recorded for every identifier.

We now define the longest path of a focused tree in a given direction. Given a tree f and a direction $\langle a\rangle$, we define the longest path as the longest cycle-free path of f compatible with the direction, i.e. that does not start in the $\langle\overline{a}\rangle$ direction. By definition of the trees, if $\langle a\rangle$ is $\langle 1\rangle$ or $\langle 2\rangle$, then the path is only made of $\langle 1\rangle$ and $\langle 2\rangle$ steps. If $\langle a\rangle$ is $\langle\overline{1}\rangle$ or $\langle\overline{2}\rangle$, then the path is a sequence of $\langle\overline{1}\rangle$ or $\langle\overline{2}\rangle$ steps followed by a sequence of $\langle 1\rangle$ and $\langle 2\rangle$ steps joined by either a $\langle\overline{1}\rangle\langle 2\rangle$ or a $\langle\overline{2}\rangle\langle 1\rangle$ sequence.

We may now prove the property that f belongs to the interpretation of a finite unfolding ψ of φ by progressively building it, relying on an induction on the lexical order of:

1. the number of identifiers in ψ not yet annotated with a direction and an integer;
2. the sum of the integers of every annotated identifier in ψ;
3. the size of ψ.

The base cases are for the true formula, the atomic proposition and its negation, the start proposition and its negation, and the negation of the existential formula. The result is immediate for all these results as they do not involve recursive formulas.

For the inductive case, we proceed by case on the syntax of ψ. The interesting cases are recursive formulas (in every other case, the size of the formula decreases while leaving the other induction metrics unchanged as annotations are updated only when unfolding formulas). In the case of a formula involving unannotated identifiers, they become annotated (thus decreasing the number of unannotated identifiers) and associated to the size of the tree, and we conclude by induction. In the case of an annotated formula recursion $\psi = \mu\overline{X_i.\varphi_i}$ in φ_j, this formula may only have been produced by a previous expansion where X_j was replaced by ψ. As the formula is cycle-free, at least one modality has been encountered and it was annotated by X_j, since X_j was free in the formula before the previous expansion. Moreover, every modality encountered since the previous unfolding was also annotated by X_j, and as the formula is cycle-free these modalities are all compatible. Thus the longest path of f in X_j's direction has decreased by at least one, and as the other identifiers may only have decreased, after expansion the sum has decreased, and we conclude by induction. □

In the rest of the chapter, we only consider least fixpoints. An important consequence of Lemma 4.3.6 is that the logic restricted in this way is closed under negation using De Morgan's dualities, extended to eventualities and fixpoints as follows:

$$\neg \langle a \rangle \varphi \stackrel{\text{def}}{=} \neg \langle a \rangle \top \vee \langle a \rangle \neg\varphi$$

$$\neg\mu\overline{X_i.\varphi_i} \text{ in } \psi \stackrel{\text{def}}{=} \mu\overline{X_i.\neg\varphi_i\{\overline{X_i}/_{\neg X_i}\}} \text{ in } \neg\psi\{\overline{X_i}/_{\neg X_i}\}$$

4.4 Satisfiability-Testing Algorithm

In this section we present our algorithm, show that it is sound and complete, and prove a time complexity boundary. To check a formula φ, our algorithm builds satisfiable formulas out of some subformulas (and their negation) of φ, then checks whether φ was produced. We first describe how to extract the subformulas from φ.

4.4.1 Preliminary Definitions

For $\varphi = (\mu\overline{X_i.\varphi_i} \text{ in } \psi)$ we define $\exp(\varphi) \stackrel{\text{def}}{=} \psi\{\overline{\mu\overline{X_i.\varphi_i} \text{ in } X_i}/_{X_i}\}$ which denotes the formula ψ in which every occurrence of a X_i is replaced by $(\mu\overline{X_i.\varphi_i} \text{ in } X_i)$.

We define the Fisher-Ladner closure $\mathrm{cl}(\psi)$ of a formula ψ as the set of all subformulas of ψ where fixpoint formulas are additionally unwound once. Specifically, we define the relation $\rightarrow_e \subseteq \mathcal{L}_\mu \times \mathcal{L}_\mu$ as the least relation that satisfies the following:

- $\varphi_1 \wedge \varphi_2 \rightarrow_e \varphi_1$, $\varphi_1 \wedge \varphi_2 \rightarrow_e \varphi_2$
- $\varphi_1 \vee \varphi_2 \rightarrow_e \varphi_1$, $\varphi_1 \vee \varphi_2 \rightarrow_e \varphi_2$
- $\langle a \rangle \varphi' \rightarrow_e \varphi'$
- $\mu\overline{X_i.\varphi_i}$ in $\psi \rightarrow_e \exp(\mu\overline{X_i.\varphi_i}$ in $\psi)$

The closure $\mathrm{cl}(\psi)$ is the smallest set S that contains ψ and is closed under the relation $\rightarrow_e$, i.e. if $\varphi_1 \in S$ and $\varphi_1 \rightarrow_e \varphi_2$ then $\varphi_2 \in S$.

We call $\Sigma(\psi)$ the set of atomic propositions σ used in ψ along with another name, σ_x, that does not occur in ψ to represent atomic propositions not occurring in ψ.

We define $\mathrm{cl}^*(\psi) = \mathrm{cl}(\psi) \cup \{\neg\varphi \mid \varphi \in \mathrm{cl}(\psi)\}$. Every formula $\varphi \in \mathrm{cl}^*(\psi)$ can be seen as a Boolean combination of formulas of a set called the Lean of ψ, inspired from [Pan et al., 2006]. We note this set $\mathrm{Lean}(\psi)$ and define it as follows:

$$\mathrm{Lean}(\psi) = \left\{\langle a \rangle \top \mid a \in \{1, 2, \overline{1}, \overline{2}\}\right\} \cup \Sigma(\psi) \cup \{\text{Ⓢ}\} \cup \{\langle a \rangle \varphi \mid \langle a \rangle \varphi \in \mathrm{cl}(\psi)\}$$

A ψ-type (or simply a "type") (Hintikka set in the temporal logic literature) is a set $t \subseteq \mathrm{Lean}(\psi)$ such that:

- $\forall \langle a \rangle \varphi \in \mathrm{Lean}(\psi), \langle a \rangle \varphi \in t \Rightarrow \langle a \rangle \top \in t$ (modal consistency);
- $\left\langle \overline{1} \right\rangle \top \notin t \vee \left\langle \overline{2} \right\rangle \top \notin t$ (a tree node cannot be both a first child and a second child);
- exactly one atomic proposition $p \in t$; we use the function $\sigma(t)$ to return the atomic proposition of a type t;
- Ⓢ may belong to t.

We call $\mathrm{Typ}(\psi)$ the set of ψ-types. For a ψ-type t, the complement of t is the set $\mathrm{Lean}(\psi) \setminus t$.

A type determines a truth assignment of every formula in $\mathrm{cl}^*(\psi)$ with the relation $\dot{\in}$ defined in Figure 4.4. Note that such derivations are finite because the number of naked $\mu\overline{X_i.\varphi_i}$ in ψ (that do not occur under modalities) strictly decreases after each expansion.

We often write $\varphi \dot{\in} t$ if there are some T, F such that $\varphi \dot{\in} t \Longrightarrow (T, F)$. We say that a formula φ is true at a type t iff $\varphi \dot{\in} t$.

We now relate a formula to the truth assignment of its ψ-types.

$$\frac{}{\top \dot{\in} t \Longrightarrow (\emptyset, \emptyset)} \qquad \frac{\varphi \in \mathrm{Lean}(\psi) \qquad \varphi \in t}{\varphi \dot{\in} t \Longrightarrow (\{\varphi\}, \emptyset)}$$

$$\frac{\varphi_1 \dot{\in} t \Longrightarrow (T_1, F_1) \qquad \varphi_2 \dot{\in} t \Longrightarrow (T_2, F_2)}{\varphi_1 \wedge \varphi_2 \dot{\in} t \Longrightarrow (T_1 \cup T_2, F_1 \cup F_2)} \qquad \frac{\varphi_1 \dot{\in} t \Longrightarrow (T_1, F_1)}{\varphi_1 \vee \varphi_2 \dot{\in} t \Longrightarrow (T_1, F_1)}$$

$$\frac{\varphi_2 \dot{\in} t \Longrightarrow (T_2, F_2)}{\varphi_1 \vee \varphi_2 \dot{\in} t \Longrightarrow (T_2, F_2)} \qquad \frac{\varphi \dot{\notin} t \Longrightarrow (T, F)}{\neg\varphi \dot{\in} t \Longrightarrow (T, F)}$$

$$\frac{\exp(\mu\overline{X_i.\varphi_i} \text{ in } \psi) \dot{\in} t \Longrightarrow (T, F)}{\mu\overline{X_i.\varphi_i} \text{ in } \psi \dot{\in} t \Longrightarrow (T, F)} \qquad \frac{\varphi \in \mathrm{Lean}(\psi) \qquad \varphi \notin t}{\varphi \dot{\notin} t \Longrightarrow (\emptyset, \{\varphi\})}$$

$$\frac{\varphi_1 \dot{\notin} t \Longrightarrow (T_1, F_1) \qquad \varphi_2 \dot{\notin} t \Longrightarrow (T_2, F_2)}{\varphi_1 \vee \varphi_2 \dot{\notin} t \Longrightarrow (T_1 \cup T_2, F_1 \cup F_2)} \qquad \frac{\varphi_1 \dot{\notin} t \Longrightarrow (T_1, F_1)}{\varphi_1 \wedge \varphi_2 \dot{\notin} t \Longrightarrow (T_1, F_1)}$$

$$\frac{\varphi_2 \dot{\notin} t \Longrightarrow (T_2, F_2)}{\varphi_1 \wedge \varphi_2 \dot{\notin} t \Longrightarrow (T_2, F_2)} \qquad \frac{\varphi \dot{\in} t \Longrightarrow (T, F)}{\neg\varphi \dot{\notin} t \Longrightarrow (T, F)}$$

$$\frac{\exp(\mu\overline{X_i.\varphi_i} \text{ in } \psi) \dot{\notin} t \Longrightarrow (T, F)}{\mu\overline{X_i.\varphi_i} \text{ in } \psi \dot{\notin} t \Longrightarrow (T, F)}$$

Figure 4.4: Truth assignment of a formula

Proposition 4.4.1. *If $\varphi \dot{\in} t \Longrightarrow (T, F)$, then we have $T \subseteq t$, $F \subseteq Lean(\varphi) \setminus t$, and $\bigwedge_{\psi \in T} \psi \wedge \bigwedge_{\psi \in F} \neg\psi$ implies φ (every tree in the interpretation of the first formula is in the interpretation of the second). If $\varphi \dot{\notin} t \Longrightarrow (T, F)$, then we have $T \subseteq t$, $F \subseteq Lean(\varphi) \setminus t$, and $\bigwedge_{\psi \in T} \psi \wedge \bigwedge_{\psi \in F} \neg\psi$ implies $\neg\varphi$.*

Proof: Immediate by induction on the derivations. □

We next define a compatibility relation between types to state that two types are related according to a modality.

Definition 4.4.2 (Compatibility relation)**.** *Two types t and t' are compatible under $a \in \{1, 2\}$, written $\Delta_a(t, t')$, iff*

$$\forall \langle a \rangle \varphi \in Lean(\psi), \langle a \rangle \varphi \in t \Leftrightarrow \varphi \dot{\in} t'$$
$$\forall \langle \overline{a} \rangle \varphi \in Lean(\psi), \langle \overline{a} \rangle \varphi \in t' \Leftrightarrow \varphi \dot{\in} t$$

4.4.2 The Algorithm

The algorithm works on sets of triples of the form (t, w_1, w_2) where t is a type, and w_1 and w_2 are sets of types which represent every witness for t according to relations $\Delta_1(t, \cdot)$ and $\Delta_2(t, \cdot)$.

$$
\begin{aligned}
\mathtt{Upd}(X) \stackrel{\text{def}}{=} X \cup & \left\{ (t, \mathtt{w}_1(t, X^\circ), \mathtt{w}_2(t, X^\circ)) \mid \begin{array}{l} \text{Ⓢ} \notin t \subseteq \mathrm{Typ}(\psi) \\ \wedge \langle 1 \rangle \top \in t \Rightarrow \mathtt{w}_1(t, X^\circ) \neq \emptyset \\ \wedge \langle 2 \rangle \top \in t \Rightarrow \mathtt{w}_2(t, X^\circ) \neq \emptyset \end{array} \right\} \\
\cup & \left\{ (t, \mathtt{w}_1(t, X^\circ), \mathtt{w}_2(t, X^\circ))^\bullet \mid \begin{array}{l} \text{Ⓢ} \in t \subseteq \mathrm{Typ}(\psi) \\ \wedge \langle 1 \rangle \top \in t \Rightarrow \mathtt{w}_1(t, X^\circ) \neq \emptyset \\ \wedge \langle 2 \rangle \top \in t \Rightarrow \mathtt{w}_2(t, X^\circ) \neq \emptyset \end{array} \right\} \\
\cup & \left\{ (t, \mathtt{w}_1(t, X^\bullet), \mathtt{w}_2(t, X^\circ))^\bullet \mid \begin{array}{l} \text{Ⓢ} \notin t \subseteq \mathrm{Typ}(\psi) \\ \wedge \langle 1 \rangle \top \in t \Rightarrow \mathtt{w}_1(t, X^\bullet) \neq \emptyset \\ \wedge \langle 2 \rangle \top \in t \Rightarrow \mathtt{w}_2(t, X^\circ) \neq \emptyset \end{array} \right\} \\
\cup & \left\{ (t, \mathtt{w}_1(t, X^\circ), \mathtt{w}_2(t, X^\bullet))^\bullet \mid \begin{array}{l} \text{Ⓢ} \notin t \subseteq \mathrm{Typ}(\psi) \\ \wedge \langle 1 \rangle \top \in t \Rightarrow \mathtt{w}_1(t, X^\circ) \neq \emptyset \\ \wedge \langle 2 \rangle \top \in t \Rightarrow \mathtt{w}_2(t, X^\bullet) \neq \emptyset \end{array} \right\}
\end{aligned}
$$

$$
\begin{aligned}
\mathtt{w}_a(t, X) &\stackrel{\text{def}}{=} \{\mathtt{type}(x) \mid x \in X \wedge \langle \overline{a} \rangle \top \in \mathtt{type}(x) \wedge \Delta_a(t, \mathtt{type}(x))\} \\
\mathtt{type}((t, w_1, w_2)) &\stackrel{\text{def}}{=} t \\
\mathtt{FinalCheck}(\psi, X) &\stackrel{\text{def}}{=} \exists x \in X, \mathtt{dsat}(x, \psi) \wedge \forall a \in \{\overline{1}, \overline{2}\}, \langle a \rangle \top \notin \mathtt{type}(x) \\
\mathtt{dsat}((t, w_1, w_2), \psi) &\stackrel{\text{def}}{=} \psi \doteq t \vee \exists x', \mathtt{dsat}(x', \psi) \wedge (x' \in w_1 \vee x' \in w_2) \\
X^\bullet &\stackrel{\text{def}}{=} \{x \in X \mid x = (\_, \_, \_)^\bullet\} \\
X^\circ &\stackrel{\text{def}}{=} \{x \in X \mid x = (\_, \_, \_)\}
\end{aligned}
$$

Figure 4.5: Operations used by the Algorithm.

The algorithm proceeds in a bottom-up approach, repeatedly adding new triples until a satisfying model is found (i.e. a triple whose first component is a type implying the formula), or until no more triple can be added. Each iteration of the algorithm builds types representing deeper trees (in the 1 and 2 direction) with pending backward modalities that will be fulfilled at later iterations. Types with no backward modalities are satisfiable, and if such a type implies the formula being tested, then it is satisfiable. The main iteration is as follows:

$X \leftarrow \emptyset$
repeat
  $X' \leftarrow X$
  $X \leftarrow \mathtt{Upd}(X')$
  if $\mathtt{FinalCheck}(\psi, X)$ **then**
    return "ψ is satisfiable"
until $X = X'$
return "ψ is unsatisfiable"

where $X \subseteq \mathrm{Typ}(\psi) \times 2^{\mathrm{Typ}(\psi)} \times 2^{\mathrm{Typ}(\psi)}$ and the update operation $\mathtt{Upd}(\cdot)$ and success check operation $\mathtt{FinalCheck}(\cdot, \cdot)$ are defined on Figure 4.5. The update operation requires four almost identical cases to ensure that the optional mark

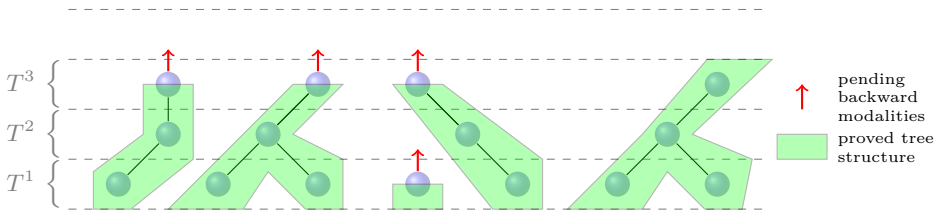


Figure 4.6: Algorithm's principle: progressive bottom-up reasoning.

remains unique. The first case corresponds to the absence of the mark, the second case to the presence of the mark at the top level, the third case to the presence of the mark deeper in the first child, and the last case to the presence of the mark deeper in the second child.

At each step of the algorithm, `FinalCheck`$(\cdot,\cdot)$ verifies whether the tested formula is implied by newly added types without pending (unproved) backward modalities, so that the algorithm may terminate as soon as a satisfying tree is found.

We note X^i the set of triples and T^i the set of types after i iterations: $T^i = \{\texttt{type}(x) \mid x \in X^i\}$. Note that T^{i+1} is the set of types for which at least one witness belongs to T^i.

4.4.3 Example Run of the Algorithm

In a sense, the algorithm performs a kind of progressive bottom-up reasoning while ensuring partial (forward) satisfiability of subformulas, as illustrated by Figure 4.6.

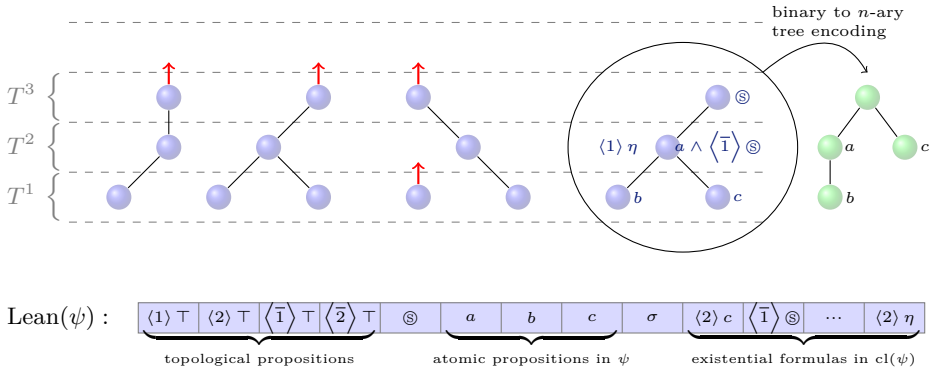
More specifically, Figure 4.7 illustrates a run of the algorithm for a sample formula ψ. Lean(ψ) is computed, and the fixpoint computation starts: the set of types T^1 contains all possible leaves. Each type added in T^i ($i \geq 2$) requires at least one witness type found in T^{i-1} (else it would have been added at some previous step $j < i$). In this example, a satisfying binary tree of depth 3 is found (as shown on Figure 4.7), therefore the algorithm stops just after computing T^3. The first XPath query is not contained in the second one: a counter-example tree is provided to the user (see Figure 4.7).

4.4.4 Correctness and Complexity

In this section we prove the correctness of the satisfiability testing algorithm, and show that its time complexity is $2^{O(|\text{Lean}(\psi)|)}$.

Theorem 4.4.3 (Correctness)**.** *The algorithm decides satisfiability of $\mathcal{L}_\mu$ formulas over finite focused trees.*

Termination. For $\psi \in \mathcal{L}_\mu$, since cl(ψ) is a finite set, Lean(ψ) and $2^{\text{Lean}(\psi)}$ are also finite. Furthermore, `Upd`$(\cdot)$ is monotonic and each X^i is included in the



$$\psi = \varphi_1 \wedge \neg\varphi_2$$
$$\varphi_1 = a \wedge (\mu Y.\, \langle 2\rangle\,(c \wedge \theta) \vee \langle 2\rangle\, Y) \wedge \langle 1\rangle\,\eta$$
$$\varphi_2 = c \wedge \underbrace{\mu X.\, \langle \overline{1}\rangle\,Ⓢ \vee \langle \overline{2}\rangle\, X}_{\theta} \wedge \langle 1\rangle\, \underbrace{\mu Z.b \vee \langle 2\rangle\, Z}_{\eta}$$

Figure 4.7: Run of the algorithm for a sample formula.

finite set $\mathrm{Typ}(\psi) \times 2^{\mathrm{Typ}(\psi)} \times 2^{\mathrm{Typ}(\psi)}$, therefore the algorithm terminates. To finish the proof, it thus suffices to prove soundness and completeness.

Preliminary Definitions for Soundness. First, we introduce a notion of partial satisfiability for a formula, where backward modalities are only checked up to a given level. A formula φ is partially satisfied iff $[\![\varphi]\!]^0_V \neq \emptyset$ as defined in Figure 4.8.

For a type t, we note $\varphi_c(t)$ its most constrained formula, where atoms are taken from Lean(ψ). In the following, $\circ$ stands for Ⓢ if Ⓢ $\in t$, and for $\neg$Ⓢ otherwise.

$$\varphi_c(t) = \sigma(t) \wedge \bigwedge_{p \in \Sigma, p \notin t} \neg p \wedge \circ \wedge \bigwedge_{\langle a\rangle \varphi \in t} \langle a\rangle\, \varphi \wedge \bigwedge_{\langle a\rangle \varphi \notin t} \neg \langle a\rangle\, \varphi$$

We now introduce a notion of paths, written ρ which are concatenations of modalities: the empty path is written ϵ, and path concatenation is written ρa.

Every path may be given a depth:

$$\begin{aligned} \mathit{depth}(\epsilon) &\stackrel{\text{def}}{=} 0 \\ \mathit{depth}(\rho a) &\stackrel{\text{def}}{=} \mathit{depth}(\rho) + 1 \quad \text{if } a \in \{1, 2\} \\ \mathit{depth}(\rho a) &\stackrel{\text{def}}{=} \mathit{depth}(\rho) - 1 \quad \text{if } a \in \{\overline{1}, \overline{2}\} \end{aligned}$$

A forward path is a path that only mentions forward modalities.

$$
\begin{aligned}
[\![\top]\!]^n_V &\stackrel{\text{def}}{=} \mathcal{F} & [\![X]\!]^n_V &\stackrel{\text{def}}{=} V(X)\\
[\![\varphi \vee \psi]\!]^n_V &\stackrel{\text{def}}{=} [\![\varphi]\!]^n_V \cup [\![\psi]\!]^n_V & [\![p]\!]^n_V &\stackrel{\text{def}}{=} \{f \mid \mathtt{nm}(f) = p\}\\
[\![\varphi \wedge \psi]\!]^n_V &\stackrel{\text{def}}{=} [\![\varphi]\!]^n_V \cap [\![\psi]\!]^n_V & [\![\neg p]\!]^n_V &\stackrel{\text{def}}{=} \{f \mid \mathtt{nm}(f) \neq p\}\\
[\![\langle \overline{1} \rangle \varphi]\!]^0_V &\stackrel{\text{def}}{=} \mathcal{F} & [\![\text{Ⓢ}]\!]^n_V &\stackrel{\text{def}}{=} \left\{f \mid f = (\sigma^{\text{Ⓢ}}[tl], c)\right\}\\
[\![\langle \overline{2} \rangle \varphi]\!]^0_V &\stackrel{\text{def}}{=} \mathcal{F} & [\![\neg\text{Ⓢ}]\!]^n_V &\stackrel{\text{def}}{=} \{f \mid f = (\sigma[tl], c)\}
\end{aligned}
$$

$$
\begin{aligned}
[\![\langle \overline{1} \rangle \varphi]\!]^{n>0}_V &\stackrel{\text{def}}{=} \left\{f\langle 1\rangle \mid f \in [\![\varphi]\!]^{n-1}_V \wedge f\langle 1\rangle \text{ defined}\right\}\\
[\![\langle \overline{2} \rangle \varphi]\!]^{n>0}_V &\stackrel{\text{def}}{=} \left\{f\langle 2\rangle \mid f \in [\![\varphi]\!]^{n-1}_V \wedge f\langle 2\rangle \text{ defined}\right\}\\
[\![\langle 1 \rangle \varphi]\!]^{n}_V &\stackrel{\text{def}}{=} \left\{f\langle \overline{1}\rangle \mid f \in [\![\varphi]\!]^{n+1}_V \wedge f\langle \overline{1}\rangle \text{ defined}\right\}\\
[\![\langle 2 \rangle \varphi]\!]^{n}_V &\stackrel{\text{def}}{=} \left\{f\langle \overline{2}\rangle \mid f \in [\![\varphi]\!]^{n+1}_V \wedge f\langle \overline{2}\rangle \text{ defined}\right\}\\
[\![\neg\langle a\rangle \top]\!]^n_V &\stackrel{\text{def}}{=} \{f \mid f\langle a\rangle \text{ undefined}\}\\
[\![\mu\overline{X_i.\varphi_i} \text{ in } \psi]\!]^n_V &\stackrel{\text{def}}{=} \text{let } T_i = \left(\bigcap\left\{\overline{T_i} \subseteq \overline{\mathcal{F}} \mid [\![\overline{\varphi_i}]\!]^n_{V[\overline{T_i/X_i}]} \subseteq \overline{T_i}\right\}\right)_i\\
&\qquad \text{in } [\![\psi]\!]^n_{V[\overline{T_i/X_i}]}
\end{aligned}
$$

Figure 4.8: Partial satisfiability

We define a tree of types $\mathcal{T}$ as a tree whose nodes are types, $\mathcal{T}(\bullet) = t$, with at most two children, $\mathcal{T}\langle 1\rangle$ and $\mathcal{T}\langle 2\rangle$. The navigation in trees of types is trivially extended to forward paths. A tree of types is consistent iff for every forward path ρ and for every child a of $\mathcal{T}\langle\rho\rangle$, we have $\overline{\mathcal{T}\langle\rho\rangle(\bullet)} = t$, $\mathcal{T}\langle\rho a\rangle(\bullet) = t'$ implies $\langle a\rangle\top \in t$, $\langle\overline{a}\rangle\top \in t'$, and $\Delta_a(t, t')$.

Given a consistent tree of types $\mathcal{T}$, we now define a dependency graph whose nodes are pairs of a forward path ρ and a formula in $t = \mathcal{T}\langle\rho\rangle(\bullet)$ or the negation of a formula in the complement of t. The directed edges of the graph are labeled with modalities consistent with the tree. This graph corresponds to what the algorithm ultimately builds, as every iteration discovers longer forward paths. For every (ρ, φ) in the nodes we build the following edges:

- $\varphi \in \Sigma(\psi) \cup \neg\Sigma(\psi) \cup \{\text{Ⓢ}, \neg\text{Ⓢ}, \langle a\rangle\top, \neg\langle a\rangle\top\}$: no edge
- $\rho = \epsilon$ and $\varphi = \langle\overline{a}\rangle\varphi'$ with $a \in \{1, 2\}$: no edge
- $\rho = \rho' a$ and $\varphi = \langle a'\rangle\varphi'$: let $t = \mathcal{T}\langle\rho\rangle(\bullet)$.

  We first consider the case where $a' \in \{1, 2\}$ and let $t' = \mathcal{T}\langle\rho a'\rangle(\bullet)$. As $\mathcal{T}$ is consistent, we have $\varphi' \dot{\in} t'$ hence there are T, F such that $\varphi' \dot{\in} t' \Longrightarrow (T, F)$ with T a subset of t', and F a subset of the complement of t'. For every $\varphi_T \in T$ we add an edge a' to $(\rho a', \varphi_T)$, and for every $\varphi_F \in F$ we add an edge a' to $(\rho a', \neg\varphi_F)$.

We now consider the case where $a' \in \{\overline{1}, \overline{2}\}$ and first show that we have $a' = \overline{a}$. As $\mathcal{T}$ is consistent, we have $\langle \overline{a} \rangle \top$ in t. Moreover, as t is a tree type, it must contain $\langle a' \rangle \top$. As a' is a backward modality, it must be equal to $\overline{a}$ as at most one may be present. Hence we have $\rho' a a' = \rho'$ and we let $t' = \mathcal{T} \langle \rho' \rangle (\bullet)$. By consistency, we have $\varphi' \dot{\in} t'$, hence $\varphi' \dot{\in} t' \Longrightarrow (T, F)$ and we add edges as in the previous case: to (ρ', φ_T) and to $(\rho', \neg\varphi_F)$.

- $\rho = \rho' a$ and $\varphi = \neg \langle a' \rangle \varphi'$: let $t = \mathcal{T} \langle \rho \rangle (\bullet)$. If $\langle a' \rangle \top$ is not in t then no edge is added. Otherwise, we proceed as in the previous case. For downward modalities, we let $t' = \mathcal{T} \langle \rho a' \rangle (\bullet)$ and we compute $\varphi' \dot{\notin} t' \Longrightarrow (T, F)$, which we know to hold by consistency. We then add edges to $(\rho a', \varphi_T)$ and to $(\rho a', \neg\varphi_F)$ as before. For upward modalities, as we have $\langle a' \rangle \top$ in t, we must have $a' = \overline{a}$ and we let $t' = \mathcal{T} \langle \rho' \rangle (\bullet)$. We compute $\varphi' \dot{\notin} t' \Longrightarrow (T, F)$ and we add the edges to (ρ', φ_T) and to $(\rho', \neg\varphi_F)$ as before.

Lemma 4.4.4. *The dependency graph of a consistent tree of types of a cycle-free formula is cycle free.*

Proof: The proof proceeds by induction on the depth of the cycle, relying on the fact that the dependency graph is consistent with the tree structure (i.e. if a 1 edge reaches a node, no $\overline{2}$ edge may leave this node). The induction case is trivial: if there is a cycle of depth n, there must be a cycle of depth $n-1$, a contradiction.

The base case is for a cycle of depth 1. We describe one case, where the cycle is $(\rho, \langle 1 \rangle \varphi) \longrightarrow^1 (\rho 1, \langle \overline{1} \rangle \psi) \longrightarrow^{\overline{1}} (\rho, \langle 1 \rangle \varphi)$. As φ must be a subformula of ψ and ψ a subformula of φ, they are both recursive formula. An analysis of the shape of φ, based on the derivations $\varphi \dot{\in} t \Longrightarrow (T, F)$ and $\psi \dot{\in} t' \Longrightarrow (T', F')$ with $\langle 1 \rangle \psi \in T$ and $\langle \overline{1} \rangle \varphi \in T'$ then shows that φ is not a cycle-free formula, a contradiction. □

Lemma 4.4.5 (Soundness)**.** *Let T be the result set of the algorithm. For any type $t \in T$ and any φ such that $\varphi \dot{\in} t$, then $[\![\varphi]\!]^0_\emptyset \neq \emptyset$.*

Proof:

The proof proceeds by induction on the number of steps of the algorithm. For every t in T^n and every witness tree $\mathcal{T}$ rooted at t built from X^n, we show that $\mathcal{T}$ is a consistent tree type and we build a focused tree f that is rooted (i.e. of the shape $(\sigma^\circ[tl], (\epsilon, \mathit{Top}, tl'))$). The tree f is in the partial interpretation of $\varphi_c(t)$: $f \langle \rho \rangle \in [\![\varphi_c(\mathcal{T} \langle \rho \rangle (\bullet))]\!]^{depth(\rho)}_\emptyset$ for any path ρ whose depth is 0 or more, and f contains the start mark only if Ⓢ occurs in $\mathcal{T}$. We then show that for all $\varphi \dot{\in} t$, we have $f \in [\![\varphi]\!]^0_\emptyset$.

The base case is trivial by the shape of t: it may only contain backward modalities (trivially satisfied at level 0), one atomic proposition, and one start proposition. Moreover there is only one tree of witnesses to consider, the tree whose only node is t. If the atomic proposition is p, then the focused tree

returned is either $(p^{Ⓢ}[\epsilon], (\epsilon, \mathit{Top}, \epsilon))$ or $(p[\epsilon], (\epsilon, \mathit{Top}, \epsilon))$ depending on the start proposition.

In the inductive case, we consider every witness types for both downward modalities, t_1 and t_2. For each of them, we consider every tree type $\mathcal{T}_1$ and $\mathcal{T}_2$ and build a tree type rooted at t which is consistent by definition of the algorithm. By induction, we have f_1 and f_2 such that $f_1 \langle\rho\rangle \in [\![\varphi_c(\mathcal{T} \langle 1\rho\rangle (\bullet))]\!]_\emptyset^{depth(\rho)}$ and $f_2 \langle\rho\rangle \in [\![\varphi_c(\mathcal{T} \langle 2\rho\rangle (\bullet))]\!]_\emptyset^{depth(\rho)}$ for any path ρ whose depth is 0 or more. If either $\mathcal{T}_1$ or $\mathcal{T}_2$ contains Ⓢ, then f_1 or f_2 contains the start mark by induction. Moreover, by definition of the algorithm, it is the case for only one of them and Ⓢ is not in t.

Let f_1 be $(\sigma_1^\circ[tl_1], (\epsilon, \mathit{Top}, tr_1))$ and f_2 be $(\sigma_2^\circ[tl_2], (\epsilon, \mathit{Top}, tr_2))$. Let f be the tree $(\sigma(t)^\circ[\sigma_1^\circ[tl_1] :: tr_1], (\epsilon, \mathit{Top}, \sigma_2^\circ[tl_2] :: tr_2))$ where $\sigma(t)^\circ$ is $\sigma(t)^{Ⓢ}$ if Ⓢ $\in t$, and $\sigma(t)$ otherwise. Note that f contains exactly one start mark iff Ⓢ $\in \mathcal{T}$.

We next show that if $f_1 \langle\rho\rangle$ is in $[\![\varphi_c(\mathcal{T} \langle 1\rho\rangle (\bullet))]\!]_\emptyset^{depth(\rho)}$, then the tree $f \langle 1\rho\rangle$ is in $[\![\varphi_c(\mathcal{T} \langle 1\rho\rangle (\bullet))]\!]_\emptyset^{depth(\rho)}$, and the same for the other modality. This is shown by induction on the depth of the path, remarking that every backward modality at level 0 is trivially satisfied.

We then proceed to show that f satisfies $\varphi_c(t)$ at level 0. To do so, we need a further induction on the dependency tree. Let ρ be a path of the dependency tree and ψ be a formula at that path in the dependency tree, we show that $f \langle\rho\rangle \in [\![\psi]\!]_V^{depth(\rho)}$. To do so, we rely on $f \langle\rho\rangle \in [\![\psi]\!]_V^{depth(\rho)-1}$ if $\mathit{depth}(\rho) \neq 0$. In the base case at depth 0, the result is by construction as the formula is either a backward modality or an atomic formula. In the base case at another depth, the case is immediate by induction as the formula has to be an atomic formula whose interpretation does not depend on the depth. In the induction case, we conclude by the inductive hypothesis and by definition of partial satisfiability.

We conclude the proof by noticing that the final selected type has no backward modality, hence $[\![\varphi_c(t)]\!]_0^\emptyset = [\![\varphi_c(t)]\!]_\emptyset$.

□

Lemma 4.4.6 (Completeness)**.** *For a cycle-free closed formula $\varphi \in \mathcal{L}_\mu$, if $[\![\varphi]\!]_\emptyset \neq \emptyset$ then the algorithm terminates with a set of triples X such that* `FinalCheck`(φ, X).

Proof: Let $f \in [\![\varphi]\!]_\emptyset$ be a smallest focused tree validating the formula such that the names occurring in f are either also occurring in φ or are a single other name σ_x. By Lemma 4.3.6, there is a finite unfolding of φ such that f belongs to its interpretation. Hence there is a finite satisfiability derivation, defined in Figure 4.9, of $f \Vdash_\epsilon \varphi$.

In the satisfiability derivation, we assume the paths are normalized ($1\overline{1} = \epsilon$). Hence every path is a concatenation of a (possibly empty) backward path ρ_b followed by a forward path ρ_f.

This derivation has the following property, immediate by induction: let f the initial focused tree, then $f' \Vdash_\rho \varphi$ implies $f' = f \langle\rho\rangle$. Hence if $f_1 \Vdash_\rho \varphi_1$ and $f_2 \Vdash_\rho \varphi_2$, then $f_1 = f_2$.

$$\frac{}{f \Vdash_\rho \top} \qquad \frac{\mathtt{nm}(f) = p}{f \Vdash_\rho p} \qquad \frac{\mathtt{nm}(f) \neq p}{f \Vdash_\rho \neg p} \qquad \frac{}{(p^{\textcircled{S}}[tl], c) \Vdash_\rho \textcircled{S}} \qquad \frac{}{(p[tl], c) \Vdash_\rho \neg\textcircled{S}}$$

$$\frac{f \Vdash_\rho \varphi}{f \Vdash_\rho \varphi \vee \psi} \qquad \frac{f \Vdash_\rho \psi}{f \Vdash_\rho \varphi \vee \psi} \qquad \frac{f \Vdash_\rho \varphi \qquad f \Vdash_\rho \psi}{f \Vdash_\rho \varphi \wedge \psi} \qquad \frac{f\,\langle 1\rangle \Vdash_{\rho 1} \varphi}{f \Vdash_\rho \langle 1\rangle\,\varphi}$$

$$\frac{f\,\langle 2\rangle \Vdash_{\rho 2} \varphi}{f \Vdash_\rho \langle 2\rangle\,\varphi} \qquad \frac{f\,\langle \overline{1}\rangle \Vdash_{\rho \overline{1}} \varphi}{f \Vdash_\rho \langle \overline{1}\rangle\,\varphi} \qquad \frac{f\,\langle \overline{2}\rangle \Vdash_{\rho \overline{2}} \varphi}{f \Vdash_\rho \langle \overline{2}\rangle\,\varphi} \qquad \frac{f\,\langle a\rangle \text{ undefined}}{f \Vdash_\rho \neg\langle a\rangle \top}$$

$$\frac{f \Vdash_\rho \exp(\mu\overline{X_i.\varphi_i} \text{ in } \psi)}{f \Vdash_\rho \mu\overline{X_i.\varphi_i} \text{ in } \psi}$$

Figure 4.9: Satisfiability relation

We next use the satisfiability derivation to construct a run of the algorithm that concludes that φ is satisfiable. We first associate each path to a type, which we then saturate (adding formulas that are true even though the satisfiability relation does not mention them at that path). We next show that every formula at a path in the satisfiability relation is implied by the type at that path, and that types are consistent according to the $\Delta_a(t, t')$ relation. We then conclude that the types are created by a run of the algorithm by induction on the paths.

More precisely, we first describe how we build t_ρ. Let Φ_ρ the set of formulas at path ρ. We first add every formula of Φ_ρ that is in Lean(φ), then we complete this set to yield a correct type: if $\langle a\rangle\,\psi \in \Phi_\rho$ then we add $\langle a\rangle \top$; for every modality a for which $f\,\langle a\rangle$ is defined we add $\langle a\rangle \top$; if there is no atomic proposition in Φ_ρ then we add $\mathtt{nm}(f\,\langle\rho\rangle)$; finally if $f\,\langle\rho\rangle$ has the start mark we add $\textcircled{S}$.

We next saturate the types. For every path t_ρ if $t_{\rho a}$ exists, if $\langle a\rangle\,\psi \in$ Lean(φ), and if $\psi \mathrel{\dot\in} t_{\rho a}$ then we add $\langle a\rangle\,\psi$ to t_ρ. This procedure is repeated until it does not change any type. Termination is a consequence of the finite size of the lean and of the number of paths. The resulting types are satisfiable as they are before saturation (since a focused tree satisfies them) and each formula added during saturation is first checked to be implied by the type.

We next show (*): for any given path ρ, if $\varphi_\rho \in \Phi_\rho$ then $\varphi_\rho \mathrel{\dot\in} t_\rho$, by induction on the satisfiability derivation. Base cases with no negation are immediate by definition of t_ρ as these are formulas of the lean. For base cases with negation, we rely on the fact that $f\,\langle\rho\rangle$ satisfies the formula, hence we cannot for instance have p and $\neg p$ in Φ_ρ. If $\neg\langle a\rangle \top \in \Phi_\rho$ then we cannot also have $\langle a\rangle\,\psi \in \Phi_\rho$ as ρa is not a valid path, hence $\langle a\rangle \top$ is not in t_ρ thus $\neg\langle a\rangle \top \mathrel{\dot\in} t_\rho$. The inductive cases of this induction (disjunction, conjunction, recursion) are immediate as they correspond to the definition of $\cdot \mathrel{\dot\in} \cdot$.

We next show that for every type t_ρ and $t_{\rho a}$ where a is a forward modality, we have $\langle\overline{a}\rangle \top \in t_{\rho a}$ and $\Delta_a(t_\rho, t_{\rho a})$. (Note that, by path normalization, the

types considered may be $t_{\overline{12}}$ and $t_{\overline{1}}$ for modality 2.) The first condition is immediate by construction of $t_{\rho a}$ as $f\langle\rho a\rangle$ is defined. For the second condition, let $\langle a\rangle\psi \in t_\rho$. If $\langle a\rangle\psi \in \Phi_\rho$, then it occurs in the satisfiability derivation with an hypothesis $f_{\rho a} \Vdash_{\rho a} \psi$. In this case we have $\psi \mathrel{\dot\in} t_{\rho a}$ by (*). If $\langle a\rangle\psi \notin \Phi_\rho$ then it was added during saturation and the result is immediate by construction. Conversely, if $\psi \mathrel{\dot\in} t_{\rho a}$ then by saturation we have $\langle a\rangle\psi \in t_\rho$. We now consider the case $\langle\overline{a}\rangle\psi \in t_{\rho a}$. The proof goes exactly as before, distinguishing the case where the formula is in $\Phi_{\rho a}$ and the case where it was added by saturation.

We now show that there is a run of the algorithm that produces these types. We proceed by induction on the paths in the downward direction: if $t_{\rho a}$ has been proven for a partial run for $a \in \{1,2\}$, then t_ρ is proven for the next step of the algorithm. Moreover, we show that $(t_\rho, \{t_{\rho 1}\}, \{t_{\rho 2}\})$ is marked iff a forward subtree of $f\langle\rho\rangle$ contains the start mark. The base case is for paths with no descendants, hence no witness is required. The algorithm then adds $(t_\rho, \emptyset, \emptyset)$ to its set of types, with a mark iff $Ⓢ \in t_\rho$, iff $f\langle\rho\rangle$ is marked.

We now consider the inductive case. By induction, a partial run of the algorithm returns $t_{\rho 1}$ and/or $t_{\rho 2}$. We first show that t_ρ is returned in the next step of the algorithm, taking these two types as witnesses. We first remark that if either witness is marked then the other is not and the mark is not at $f\langle\rho\rangle$, since there is only one start mark in f, and if the mark is at $f\langle\rho\rangle$, then neither witness is marked. For each child $a \in \{1,2\}$ we have $\Delta_a(t_\rho, t_{\rho a})$ and $\langle\overline{a}\rangle\top \in t_{\rho a}$, hence the triple (t_ρ, W_1, W_2) with $t_{\rho 1} \in W_1$ and $t_{\rho 2} \in W_2$ is added by the algorithm.

We may now conclude. At the end of the induction, the last path considered, ρ_0, has no predecessor, hence it is the longest backward only path. Since $f\langle\rho_0\rangle$ is the root of the tree, we have $\langle\overline{1}\rangle\top \notin t_{\rho_0}$ and $\langle\overline{2}\rangle\top \notin t_{\rho_0}$. Moreover, as the start mark is somewhere in f, it is in a forward subtree of $f\langle\rho_0\rangle$, hence the final type is marked. Finally, t_ϵ is in the witness tree of the final type, and since $f \Vdash_\epsilon \varphi$, we have $\varphi \mathrel{\dot\in} t_\epsilon$. □

We now present one of the main contributions of this chapter: the complexity of our algorithm is $2^{O(n)}$ where n is the formula size. It is well-known that $\mathrm{cl}(\psi)$ is a finite set and its size is linear with respect to the size of ψ (i.e., the number of operators and propositional variables appearing in ψ) [Kozen, 1983]. Therefore $|\mathrm{Lean}(\psi)|$ is also trivially linear with respect to the size of ψ.[1]

Theorem 4.4.7 (Complexity)**.** *For $\psi \in \mathcal{L}_\mu$ the satisfiability problem $[\![\psi]\!]_\emptyset \neq \emptyset$ is decidable in time $2^{O(n)}$ where $n = |Lean(\psi)|$.*

Proof: $|\mathrm{Typ}(\psi)|$ is bounded by $\left|2^{\mathrm{Lean}(\psi)}\right|$ which is $2^{O(n)}$. During each iteration, the algorithm adds at least one new type (otherwise it terminates), thus it performs at most $2^{O(n)}$ iterations. We now detail what it does at each iteration. For each type that may be added (there are $2^{O(n)}$ of them), there are two traversals of the set of types at the previous step to collect witnesses. Hence there are $2 * 2^{O(n)} * 2^{O(n)} = 2^{O(n)}$ witness tests at each iteration. Each witness

[1] The acute reader may notice that for large formulas, $|\mathrm{Lean}(\psi)|$ is usually smaller than the size of ψ since disjunctions, conjunctions and negations are not members of $\mathrm{Lean}(\psi)$.

test involves a membership test and a Δ_a test. In the implementation these are precomputed: for every formula $\langle a \rangle \varphi$ in the lean, the subsets (T, F) of the lean that must be true and false respectively for φ to be true are precomputed, so testing $\varphi \dot{\in} t$ are simple inclusion and disjunction tests. The `FinalCheck` condition test at most $2^{O(n)}$ ψ-types and each test takes at most $2^{O(n)}$ (testing the formulas containing Ⓢ against ψ). Therefore, the worst case global time complexity of the algorithm does not exceed $2^{O(n)}$. □

4.5 Implementation Techniques

This section describes the main techniques used for implementing an effective $\mathcal{L}_\mu$ decision procedure. Our implementation is publicly available and usable through a web interface [Genevès & Layaïda, 2006a].

4.5.1 Implicit Representation of Sets of ψ-Types

Our implementation relies on a symbolic representation and manipulation of sets of ψ-types using Binary Decision Diagrams (BDDs) [Bryant, 1986]. BDDs provide a canonical representation of Boolean functions. Experience has shown that this representation is very compact for very large Boolean functions. Their effectiveness is notably well known in the area of formal verification of systems [Edmund M. Clarke et al., 1999].

First, we observe that the implementation can avoid keeping track of every possible witnesses of each ψ-type. In fact, for a formula φ, we can test $[\![\varphi]\!]_\emptyset \neq \emptyset$ by testing the satisfiability of the (linear-size) "plunging" formula $\psi = \mu X.\varphi \vee \langle 1 \rangle X \vee \langle 2 \rangle X$ at the root of focused trees. That is, checking $[\![\psi]\!]^0_\emptyset \neq \emptyset$ while ensuring there is no unfulfilled upward eventuality at top level 0. One advantage of proceeding this way is that the implementation only need to deal with a current set of ψ-types at each step.

We now introduce a bit-vector representation of ψ-types. Types are complete in the sense that either a subformula or its negation must belong to a type. It is thus possible for a formula $\varphi \in \text{Lean}(\psi)$ to be represented using a single BDD variable. For $\text{Lean}(\psi) = \{\varphi_1, ..., \varphi_m\}$, we represent a subset $t \subseteq \text{Lean}(\psi)$ by a vector $\vec{t} = \langle t_1, ..., t_m \rangle \in \{0, 1\}^m$ such that $\varphi_i \in t$ iff $t_i = 1$. A BDD with m variables is then used to represent a set of such bit vectors.

We define auxiliary predicates for programs $a \in \{1, 2\}$:

- $\text{isparent}_a(\vec{t})$ is read "$\vec{t}$ is a parent for program a" and is true iff the bit for $\langle a \rangle \top$ is true in $\vec{t}$
- $\text{ischild}_a(\vec{t})$ is read "$\vec{t}$ is a child for program a" and is true iff the bit for $\langle \overline{a} \rangle \top$ is true in $\vec{t}$

For a set $T \subseteq 2^{\text{Lean}(\psi)}$, we note χ_T its corresponding characteristic function.

Encoding $\chi_{\mathrm{Typ}(\psi)}$ is straightforward with the previous definitions. We define the equivalent of $\dot{\in}$ on the bit vector representation:

$$\mathrm{status}_\varphi(\vec{t}) \stackrel{\text{def}}{=} \begin{cases} t_i & \text{if } \varphi \in \mathrm{Lean}(\psi) \\ \mathrm{status}_{\varphi'}(\vec{t}) \wedge \mathrm{status}_{\varphi''}(\vec{t}) & \text{if } \varphi = \varphi' \wedge \varphi'' \\ \mathrm{status}_{\varphi'}(\vec{t}) \vee \mathrm{status}_{\varphi''}(\vec{t}) & \text{if } \varphi = \varphi' \vee \varphi'' \\ \neg\mathrm{status}_{\varphi'}(\vec{t}) & \text{if } \varphi = \neg\varphi' \\ \mathrm{status}_{\exp(\varphi)}(\vec{t}) & \text{if } \varphi = \mu\overline{X_i.\varphi_i} \text{ in } \psi \end{cases}$$

We note $a \to b$ the implication and $a \leftrightarrow b$ the equivalence of two Boolean formulas a and b over vector bits. We can now construct the BDD of the relation Δ_a for $a \in \{1, 2\}$.

This BDD relates all pairs $(\vec{x}, \vec{y})$ that are consistent w.r.t the program a, i.e., such that $\vec{y}$ supports all of $\vec{x}$'s $\langle a \rangle \varphi$ formulas, and vice-versa $\vec{x}$ supports all of $\vec{y}$'s $\langle \overline{a} \rangle \varphi$ formulas:

$$\Delta_a(\vec{x}, \vec{y}) \stackrel{\text{def}}{=} \bigwedge_{1 \le i \le m} \begin{cases} x_i \leftrightarrow \mathrm{status}_\varphi(\vec{y}) & \text{if } \varphi_i = \langle a \rangle \varphi \\ y_i \leftrightarrow \mathrm{status}_\varphi(\vec{x}) & \text{if } \varphi_i = \langle \overline{a} \rangle \varphi \\ \top \quad \text{otherwise} \end{cases}$$

For $a \in \{1, 2\}$, we define the set of witnessed vectors:

$$\chi_{\mathtt{Wit}_a(T)}(\vec{x}) \stackrel{\text{def}}{=} \mathrm{isparent}_a(\vec{x}) \to \exists \vec{y} \ [\ h(\vec{y}) \wedge \Delta_a(\vec{x}, \vec{y})\ ]$$

where $h(\vec{y}) = \chi_T(\vec{y}) \wedge \mathrm{ischild}_a(\vec{y})$.

Then, the BDD of the fixpoint computation is initially set to the false constant, and the main function $\mathtt{Upd}(\cdot)$ is implemented as:

$$\chi_{\mathtt{Upd}(T)}(\vec{x}) \stackrel{\text{def}}{=} \chi_T(\vec{x}) \vee \left(\chi_{\mathrm{Typ}(\psi)}(\vec{x}) \wedge \bigwedge_{a \in \{1,2\}} \chi_{\mathtt{Wit}_a(T)}(\vec{x}) \right)$$

Finally, the solver is implemented as iterations over the sets $\chi_{\mathtt{Upd}(T)}$ until a fixpoint is reached. The final satisfiability condition consists in checking whether ψ is present in a ψ-type of this fixpoint with no unfulfilled upward eventuality:

$$\exists \vec{t} \left[\chi_T(\vec{t}) \wedge \bigwedge_{a \in \{1,2\}} \neg\mathrm{ischild}_a(\vec{t}) \wedge \mathrm{status}_\psi(\vec{t}) \right]$$

.

4.5.2 Satisfying Model Reconstruction

The implementation keeps a copy of each intermediate set of types computed by the algorithm, so that whenever a formula is satisfiable, a minimal satisfying model can be extracted. The top-down (re)construction of a satisfying model starts from a root (a ψ-type for which the final satisfiability condition holds), and repeatedly attempts to find successors. In order to minimize model size, only required left and right branches are built. Furthermore, for minimizing the maximal depth of the model, left and right successors of a node are successively searched in the intermediate sets of types, in the order they were computed by the algorithm.

4.5.3 Conjunctive Partitioning and Early Quantification

The BDD-based implementation involves computations of relational products of the form:

$$\exists \vec{y} \ [\ h(\vec{y}) \wedge \Delta_a(\vec{x}, \vec{y})\ ] \tag{4.1}$$

It is well-known that such a computation may be quite time and space consuming, because the BDD corresponding to the relation Δ_a may be quite large.

One famous optimization technique is conjunctive partitioning [Edmund M. Clarke et al., 1999] combined with early quantification [Pan et al., 2006]. The idea is to compute the relational product without ever building the full BDD of the relation Δ_a. This is possible by taking advantage of the form of Δ_a along with properties of existential quantification. By definition, Δ_a is a conjunction of n equivalences relating $\vec{x}$ and $\vec{y}$ where n is the number of $\langle b \rangle \varphi$ formulas in Lean(ψ) where $\varphi \neq \top$ and $b \in \{a, \overline{a}\}$:

$$\Delta_a(\vec{x}, \vec{y}) = \bigwedge_{i=1}^{n} R_i(\vec{x}, \vec{y})$$

If a variable y_k does not occur in the clauses $R_{i+1}, ..., R_n$ then the relational product (4.1) can be rewritten as:

$$\exists_{y_1, ..., y_{k-1}, y_{k+1}, ..., y_m} \left[\exists y_k \left[h(\vec{y}) \wedge \textstyle\bigwedge_{1 \leq j \leq i} R_j(\vec{x}, \vec{y}) \right] \wedge \textstyle\bigwedge_{i+1 \leq l \leq n} R_l(\vec{x}, \vec{y}) \right]$$

This allows to apply existential quantification on intermediate BDDs and thus to compose smaller BDDs. Of course, there are many ways to compose the $R_i(\vec{x}, \vec{y})$. Let ρ be a permutation of $\{0, ..., n-1\}$ which determines the order in which the partitions $R_i(\vec{x}, \vec{y})$ are combined. For each i, let D_i be the set of variables y_k with $k \in \{1, ..., m\}$ that $R_i(\vec{x}, \vec{y})$ depends on. We define E_i as the set of variables contained in $D_{\rho(i)}$ that are not contained in $D_{\rho(j)}$ for any j larger than i:

$$E_i = D_{\rho(i)} \setminus \bigcup_{j=i+1}^{n-1} D_{\rho(j)}$$

The E_i are pairwise disjoint and their union contains all the variables. The relational product (4.1) can be computed by starting from:

$$h_1(\vec{x}, \vec{y}) = \exists_{y_k \in E_0} [\ h(\vec{y}) \wedge R_{\rho(0)}(\vec{x}, \vec{y})\ ]$$

and successively computing h_{p+1} defined as follows:

$$h_{p+1}(\vec{x}, \vec{y}) = \begin{cases} \exists_{y_k \in E_p} [\ h_p(\vec{x}, \vec{y}) \wedge R_{\rho(p)}(\vec{x}, \vec{y})\ ] & \text{if } E_p \neq \emptyset \\ h_p(\vec{x}, \vec{y}) \wedge R_{\rho(p)}(\vec{x}, \vec{y}) & \text{if } E_p = \emptyset \end{cases}$$

until reaching h_n which is the result of the relational product. The ordering ρ determines how early in the computation variables can be quantified out.

This directly impact the sizes of BDDs constructed and therefore the global efficiency of the decision procedure. It is thus important to choose ρ carefully. The overall goal is to minimize the size of the largest BDD created during the elimination process. We use a heuristic taken from [Edmund M. Clarke et al., 1999] which seems to provide the best approximation and in practice has the best performance. It defines the cost of eliminating a variable y_k as the sum of the sizes of all the D_i containing y_k:

$$\sum_{1 \leq i \leq n, y_k \in D_i} |D_i|$$

The ordering ρ on the relations R_i is then defined in such a way that variables can be eliminated in the order given by a greedy algorithm which repeatedly eliminates the variable of minimum cost.

4.5.4 BDD Variable Ordering

The cost of BDD operations is very sensitive to variable ordering. Finding the optimal variable ordering is known to be NP-complete [Hojati et al., 1996], however several heuristics are known to perform well in practice [Edmund M. Clarke et al., 1999]. Choosing a good initial order of Lean(ψ) formulas does significantly improve performance. We found out that preserving locality of the initial problem is essential. Experience has shown that the variable order determined by the breadth-first traversal of the formula ψ to solve, which keeps sister subformulas in close proximity, yields better results in practice.

4.5.5 Online Implementation

The system has been implemented as a web application. Interaction with the system is offered through a user interface in a web browser. The tool is available online from:

`http://wam.inrialpes.fr/websolver/`

A screenshot of the interface is given in Figure 4.10. The user can either enter a formula through area (1) of Figure 4.10 or select from pre-loaded analysis tasks offered in area (4) of Figure 4.10. The level of details displayed by the solver can be adjusted in area (2) of Figure 4.10 and makes it possible to inspect logical translations and statistics on problem size and the different operation costs. The results of the analysis are displayed in area (3) of Figure 4.10 together with counter-examples.

4.6 Examples and Experiments

In this section we report on practical experiments that we have made using the solver implementation. These experiments can be tried with the online implementation described above.

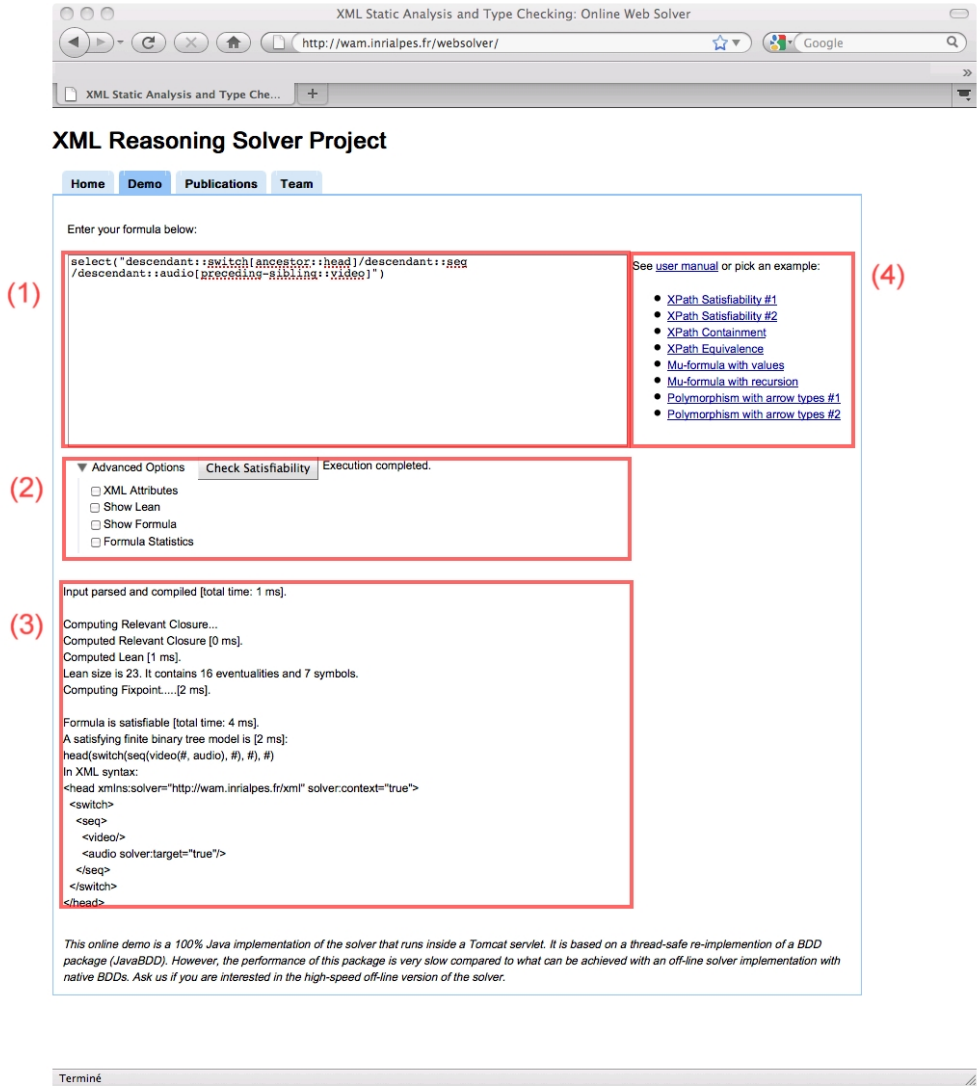


Figure 4.10: Screenshot of the Solver Interface.

4.6.1 Regular Language Equivalence

As a first, simple, application, we show how we can use our solver to decide the equivalence of regular languages. To this end, we translate regular expressions in formulas of our tree logic. The translation presented here is very naive; the actual translation used in the online implementation (using the `reg_exp` keyword) results in more concise formulas.

As illustrated in Section 4.2, our model does not allow the empty tree. To translate regular expressions which may recognize the empty word, we add a final letter e at the end of the expression. We also need to be careful with the repetition of regular expressions, of the form R^*, if R is nullable (it accepts

the empty word ϵ). A direct translation in this case would result in a recursion variable appearing naked (i.e., without a surrounding modality). We thus extract the non null part of R, written $R_{\overline{\epsilon}}$, and translate R^* as $R^*_{\overline{\epsilon}}$. We first recall how to naively extract from a regular expression R its nullable part (either ϵ or $\emptyset$), written R_ϵ, and its non null part, written $R_{\overline{\epsilon}}$.

$$\begin{aligned}
\epsilon_\epsilon &= \epsilon & \epsilon_{\overline{\epsilon}} &= \emptyset \\
a_\epsilon &= \emptyset & a_{\overline{\epsilon}} &= a \\
(R.R')_\epsilon &= R_\epsilon.R'_\epsilon & (R.R')_{\overline{\epsilon}} &= R_\epsilon.R'_{\overline{\epsilon}} \vee R_{\overline{\epsilon}}.R'_\epsilon \vee R_{\overline{\epsilon}}.R'_{\overline{\epsilon}} \\
(R^*)_\epsilon &= \epsilon & (R^*)_{\overline{\epsilon}} &= (R_{\overline{\epsilon}})^+ \\
(R \vee R')_\epsilon &= R_\epsilon \vee R'_\epsilon & (R \vee R')_{\overline{\epsilon}} &= R_{\overline{\epsilon}} \vee R'_{\overline{\epsilon}}
\end{aligned}$$

The translation of a regular expression R with a continuation c is written $[\![R]\!]_c$ and is defined as follows.

$$\begin{aligned}
[\![a]\!]_c &= a \wedge \langle 1 \rangle\, c \\
[\![\epsilon]\!]_c &= c \\
[\![R.R']\!]_c &= [\![R]\!]_{[\![R']\!]_c} \\
[\![R^*]\!]_c &= \mu x.c \vee [\![R_{\overline{\epsilon}}]\!]_x && \text{if } R_{\overline{\epsilon}} \neq \emptyset \\
[\![R^*]\!]_c &= c && \text{if } R_{\overline{\epsilon}} = \emptyset \\
[\![R \vee R']\!]_c &= [\![R]\!]_c \vee [\![R']\!]_c
\end{aligned}$$

Given a regular expression R, we translate it in our logic as the formula $[\![R]\!]_e$.

The expression `F` iff `G` is syntactic sugar for (`F`&`G`)|(¬`F`&¬`G`). It is satisfied if a tree can be found that has a node where either both `F` and `G` are true, or where neither is. The negation of this formula is satisfied if there is a tree with a node where either `F`&¬`G` or ¬`F`&`G` is true. It is unsatisfied if for every tree and for every node, either `F`&`G` is true or ¬`F`&¬`G` is true: the formulas are equivalent (they will always select the same set of nodes).

To check the equivalence of two regular expressions R_1 and R_2, we thus ask the solver the question $\neg([\![R_1]\!]_e$ iff $[\![R_2]\!]_e)$. If the formula is unsatisfied, the languages are equivalent. If it is satisfied, the solver will return a word in one language and not the other.

We illustrate this translation with several examples. To show that the languages $(ab)^*a$ and $a(ba)^*$ are equivalent, we run the following query in the solver. (This code may be copied and pasted directly in the online demo.)

```
~
(let $X = (a & <1>e) | a & <1>(b & <1> $X) in $X)
<=>
(a & <1> (let $X = e | b & <1>(a & <1> $X) in $X))
```

We now show the more complex language equivalence $(a|b)^* = a^*(ba^*)^*$. To this end, we need to circumvent a shortcoming of the solver. In the following expressions, some fixpoint variables are not syntactically guarded, yet the is no fixpoint expansions that would continue indefinitely without encountering a guard. More precisely, when unfolding a `$Z`, there will always be a `<1>` modality before reaching `$Z` again. As the more precise analysis of guardedness and cycle-freeness (see Figure 4.3) is not yet implemented, we annotate these recursive calls with `<0>` to let the solver know we guarantee these calls are guarded.

```
~
((let $X = e | (a & <1>$X) | (b & <1>$X) in $X)
<=>
(let $X = <0>$Y | (a & <1>$X),
     $Y = e | (b & <1>$Z),
     $Z = <0>$Y | (a & <1>$Z)
in $X))
```

Finally, we show that if the languages are not equivalent, the solver provides a counter-example: a word that is in one language but not the other. For instance, let us compare $(a|b)^*$ and $a^*(ba^*)^*b$. We thus try the following formula.

```
~
((let $X = e | (a & <1>$X) | (b & <1>$X) in $X)
<=>
(let $X = <0>$Y | (a & <1>$X),
     $Y = (b & <1>e) | (b & <1>$Z),
     $Z = <0>$Y | (a & <1>$Z)
in $X))
```

To which the solver replies "Formula is satisfiable [total time: 4 ms]. A satisfying finite binary tree model is [2 ms]: e." Indeed, the empty word is accepted by the first formula but not by the second.

We now extend our simple translation to study the equivalence of languages of Kleene Algebra with Test (KAT) [Kozen, 1997].

To translate KAT formulas, we add boolean propositions, written $\_p$, to the logic. We also extend our model, annotating every node with the set of propositions which are true at that node. These extensions carry over to the algorithm straightforwardly, as we only need to add to the lean the set of boolean propositions mentioned in the formula. This extension has in fact been implemented in our solver and can be tried in the online version.

We extend our translation with a case for boolean propositions $\_p$

$$[\![\_p]\!]_c = \_p \wedge c$$
$$[\![\neg\_p]\!]_c = \neg\_p \wedge c$$

Note that, unlike the translation for letters, we do not move to the next letter. One may thus specify several propositions that must concurrently be true.

As an example, consider the usual encoding of a while loop in KAT.

$$[\![(\_pa)^*\neg\_p]\!]_e = \mu x.(\neg\_p \wedge e) \vee (\_p \wedge a \wedge \langle 1\rangle\, x)$$

We can thus use the solver to test for language equality or inequality. For instance, one may show that $\_bq^*$ is different from $(\_bq)^*$ as follows.

```
~
(_b & (let $X = e | q & <1>$X in $X)
<=>
let $X = e | _b & q & <1>$X in $X)
```

The satisfying word found is `e ~_b`, which is the empty word annotated with the negation of `_b`: it belongs to the second language but not to the first.

4.6.2 Applications to XPath typing

Another natural application of the tree logic consists in the static analysis of programs that manipulate XML documents seen as trees. Backward modalities naturally capture XPath expressions that navigate upward in the tree in a succinct manner. The translations of XPath and XML type expressions into the logic are recalled from [Genevès & Layaïda, 2006c] in appendix to make the article self-contained. We also give the semantics of XPath in terms of focused trees, and prove that the generated formulas are cycle-free. Owing to these translations, we can formulate several decision problems involving XPath expressions $e_1, ..., e_n$ and XML type expressions $T_1, ..., T_n$, for which the solver provides a decision procedure. In particular, the following basic problems are of special interest:

- XPath containment: $E^{\rightarrow}[\![e_1]\!]_{[\![T_1]\!]} \wedge \neg E^{\rightarrow}[\![e_2]\!]_{[\![T_2]\!]}$ (if the formula is unsatisfiable then all nodes selected by e_1 under type constraint T_1 are selected by e_2 under type constraint T_2)
- XPath emptiness: $E^{\rightarrow}[\![e_1]\!]_{[\![T_1]\!]}$
- XPath overlap: $E^{\rightarrow}[\![e_1]\!]_{[\![T_1]\!]} \wedge E^{\rightarrow}[\![e_2]\!]_{[\![T_2]\!]}$
- XPath coverage: $E^{\rightarrow}[\![e_1]\!]_{[\![T_1]\!]} \wedge \bigwedge_{2\leq i\leq n} \neg E^{\rightarrow}[\![e_i]\!]_{[\![T_i]\!]}$
- Static type checking of an annotated XPath expression: $E^{\rightarrow}[\![e_1]\!]_{[\![T_1]\!]} \wedge \neg[\![T_2]\!]$ (if the formula is unsatisfiable then all nodes selected by e_1 under type constraint T_1 are included in the type T_2.)
- XPath equivalence under type constraints: $E^{\rightarrow}[\![e_1]\!]_{[\![T_1]\!]} \wedge \neg E^{\rightarrow}[\![e_2]\!]_{[\![T_2]\!]}$ and $\neg E^{\rightarrow}[\![e_1]\!]_{[\![T_1]\!]} \wedge E^{\rightarrow}[\![e_2]\!]_{[\![T_2]\!]}$ (This test can be used to check that the nodes selected after a modification of a type T_1 by T_2 and an XPath expression e_1 by e_2 are the same, typically when

e_1 $/a[.//b[c/*//d]/b[c//d]/b[c/d]]$
e_2 $/a[.//b[c/*//d]/b[c/d]]$

e_3 a/b//c/foll-sibling::d/e
e_4 a/b//d[prec-sibling::c]/e
e_5 a/c/following::d/e
e_6 a/b[//c]/following::d/e ∩ a/d[preceding::c]/e

e_7 *//switch[ancestor::head]//seq//audio[prec-sibling::video]

e_8 descendant::a[ancestor::a]
e_9 /descendant::*
e_{10} html/(head + body)
e_{11} html/head/descendant::*
e_{12} html/body/descendant::*

Figure 4.11: XPath Expressions Used in Experiments.

DTD	Symbols	Binary Type Variables
SMIL 1.0	19	11
XHTML 1.0 Strict	77	325

Table 4.1: Types Used in Experiments.

an input type changes and the corresponding XPath expression has to change as well.)

We carried out extensive tests with the implementation[2] [Genevès & Layaïda, 2006a] , and present here only a representative sample that includes the most complex language features such as recursive forward and backward axes, intersection, large and very recursive types with a reasonable alphabet size. The tests use XPath expressions shown on Figure 4.11 (where "//" is used as a shorthand for "/desc-or-self::*/") and XML types shown on Table 4.1. Table 4.2 presents some decision problems and corresponding performance results. Times reported in milliseconds correspond to the running time of the satisfiability solver without the (negligible) time spent for parsing and translating into $\mathcal{L}_\mu$.

The first XPath containment instance was first formulated in [Miklau & Suciu, 2004] as an example for which the proposed tree pattern homomorphism technique is incomplete. The e_8 example shows that the official XHTML DTD does not syntactically prohibit the nesting of anchors. For the XHTML case, we observe that the time needed is more important, but it remains practically relevant, especially for static analysis operations performed only at compile-time.

[2]Experiments have been conducted with a JAVA implementation running on a Pentium 4, 3 Ghz, with 512Mb of RAM with Windows XP.

XPath Decision Problem	XML Type	Time (ms)
$e_1 \subseteq e_2$ and $e_2 \not\subseteq e_1$	none	353
$e_4 \subseteq e_3$ and $e_4 \subseteq e_3$	none	45
$e_6 \subseteq e_5$ and $e_5 \not\subseteq e_6$	none	41
e_7 is satisfiable	SMIL 1.0	157
e_8 is satisfiable	XHTML 1.0	2630
$e_9 \subseteq (e_{10} \cup e_{11} \cup e_{12})$	XHTML 1.0	2872

Table 4.2: Some Decision Problems and Corresponding Results.

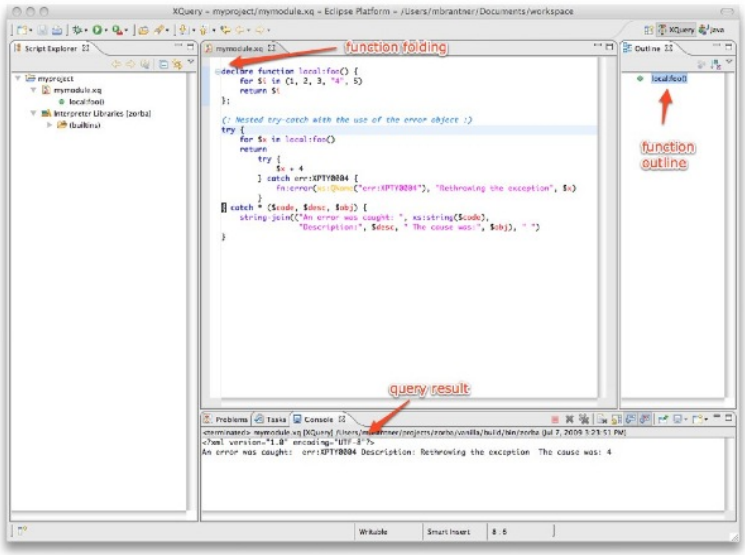


Figure 4.12: Overview of XQDT user interface.

4.7 Augmented IDE

As a proof of concept, we have integrated this static analysis inside an IDE: we have equipped the XQuery Development Toolkit (XQDT) [XQDT, 2010] with capabilities of statically detecting inconsistent paths. XQDT is a plugin for the Eclipse environment that provides support for XQuery 1.1. In particular XQDT provides code completion and code templates, as-you-type validation, and integration with existing XQuery evaluation engines. A screenshot of XQDT is given in Figure 4.12.

4.7.1 Integration Principle

We have developed a plugin extension that takes an XPath expression e and a schema S as parameters and checks for the inconsistency of e in the presence of S. The analysis functions mark inconsistent path with syntax coloring capabilities offered by the IDE plugin. For this to be possible, the IDE plugin interacts with the plugin extension in the following manner:

1. the abstract syntax tree of the program is first analyzed in order to identify XPath expressions;

2. the evaluation context of each XPath expression is built: because some XPath host languages (like XSLT or XQuery) allow variables to be defined and then used in paths, this step is necessary for correctly replacing variables occurring in paths by their definition;

3. when the static verification is triggered, each XPath expression and its evaluation context as well as the schema chosen by the programmer are transmitted to the plugin extension;

4. once the analysis is performed the plugin extension returns information (line number, character index) in order to mark inconsistent paths in the user interface.

4.7.2 Enriched Programming Experience

From the programmer's point of view, the augmented XQDT plugin can be used just as the usual XQDT plugin. The only difference happens when a given XQuery program is opened through the user interface. Two new buttons are then offered to the programmer. The first one allows him to choose a given schema (notice that this is optional: by default no schema is assumed). The second button allows the programmer to trigger the static analysis of paths which marks inconsistent XPath expressions, in the same manner as badly typed Java statements are marked in the classic Eclipse environment for editing Java programs.

The user interface of the plugin extension is shown in Figure 4.13. The screenshot shows an XQuery example where an XPath expression is automatically identified and marked as inconsistent (independently of any schema). In this case, the XPath expression "`$r/parent::book/author`" is trivially judged inconsistent by the analysis since if we replace "`$r`" by its definition we obtain an expression of the form:

```
//reviews/review/book/parent::book/author
```

that at some point attempts to navigate from a node labeled "`review`" to children nodes labeled "`book`" and then going back to the parent node labeled "`book`", which contradicts the previous steps according to which this parent node is labeled "`review`". For this reason, evaluating this path always yields an empty set of nodes, even independently from any schema. General path inconsistencies are not so trivial to detect, especially those involving schema information. This is why inconsistent paths are clearly marked à la Eclipse in the user interface: they are underlined in red and marked with red icons both in the left gutter and next to the scroll bar on the right in order to inform the programmer.

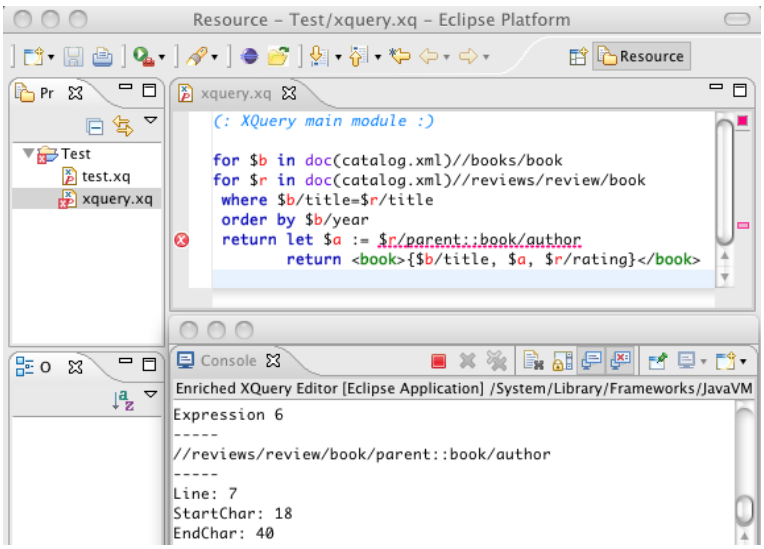


Figure 4.13: Static Analysis of Paths in Action.

4.8 Dead-Code Analysis for XQuery

In this section, we present a static analysis of XQuery programs, based on the detection of inconsistent paths (with and without schema information) in order to automatically detect and eliminate dead code. We first introduced XQuery programs, and in particular the fragment that we consider, and then the analysis of dead code.

4.8.1 XQuery Programs

An XQuery program basically takes one (or possibly several) XML document as input, performs some computation based on its tree view, and finally outputs a result in the form of another XML document. The core of the XQuery language is composed of XPath expressions that make it possible to navigate in the document tree and extract nodes that satisfy some conditions. For instance, a simplistic XQuery program is:

```
<ul>
 {
  for $x in /descendant::book return
    if $x/year>2008 then <li>$x/title<li> else ()
 }
</ul>
```

where the `for` loop uses the XPath expression `/descendant::book` that traverses the whole input XML document looking for `book` elements. The `for` loop iterates over all these elements, and for each of them, returns the value of

```
e      ::=
           ()
       |   e,e
       |   <tag>e</tag>
       |   element{e}{e}
       |   if e then e else e
       |   for $var in e (where e)? return e
       |   let $var := e (where e)? return e
       |   (some | every) $var in e satisfies e
       |   typeswitch (e) cases
       |   $var
       |   path
       |   $var/path
       |   /path
       |   e op e
       |   f(e, ..., e)
       |   arithmeticExpr
cases  ::=
           default return e
       |   case Type return e cases
```

Figure 4.14: Syntax of XQuery Programs.

the `title` subelement, provided the year is greater than 2008. Executing this program produces an XML tree as output, whose root element is named "`ul`", and whose content is populated by the execution of the loop, that creates an XHTML-like list of book titles published after 2008.

In the remaining, we consider a fragment of the XQuery programming language, whose syntax is given in Figure 4.14 (the semantics is described in [Boag et al., 2006]). This fragment focuses on the core aspects of XQuery and in particular XPath expressions for navigating and extracting information from XML trees. The syntax of XPath expressions we consider is given in Figure 4.17. We consider all XPath features for navigating forward, backward and recursively through nodes of the document. Furthermore, at each step in the navigation the selected nodes can be filtered using qualifiers, that are boolean expression between brackets that can test the existence or absence of paths. All major features of XPath are supported except general counting and general comparisons between data values (that are known to make XPath satisfiability undecidable [Benedikt et al., 2005]).

4.8.2 XQuery Dead Code

In order to illustrate the practical relevance of our approach, consider the following XQuery program:

```
<para>
  {
    for $x in //body//switch
    where $x/animateMotion
    return $x/*
```

```
  }
</para>
```

It is intended to be evaluated over SMIL[3] documents. Specifically, it has been written against the schema defining SMIL 1.0 documents. When applied to such a document, it returns all children of `switch` elements that have at least one `animateMotion` child, wrapped in a `para` element.

This code portion may be reused in the context of SMIL 2.0 documents. However, in contrast to SMIL 1.0, the occurrence of `animateMotion` is not permitted as a chid of `switch` in SMIL 2.0. In this case, the XPath expression in the `where` clause is inconsistent, and therefore the whole `for` loop is dead code. We explain how we make this static analysis automatic for a given XQuery program and a given schema in the next subsections.

4.8.3 Dead-Code Analysis

We consider a given XQuery program P and a schema S that describes constraints over the set of documents that can serve as input to P. For each XPath expression occurring in P, we check whether it is inconsistent in the presence of S. In that case, we know statically that there is no need to evaluate the path at runtime. Furthermore, we also know that all XQuery instructions that depend on this path (dead code) may be removed.

This analysis is sound and complete over the XPath navigational fragment shown in Figure 4.17. In order for the analysis to scale to programs with more complex features, we make several conservative approximations. First, we abstract over XPath features that make satisfiability undecidable (such as data value comparisons). Second, we consider that XPath expressions return sets of nodes (as in XPath 1.0) instead of node sequences (as in XPath 2.0 and XQuery). These approximations preserve soundness of our approach (if dead code is detected, it can be safely eliminated as this is really dead code). However, the analysis may be incomplete due to undecidable features, that may prevent from finding some evil dead code.

4.8.4 Static Code Refactoring

Each path which is found inconsistent indicates dead code. We perform a code dependency analysis that propagates this information in order to detect and eliminate dead code from an XQuery program. The analysis consists of inference rules of the form:

$$\frac{H}{S, \Gamma : \underline{e} \longrightarrow \underline{e}'}$$

Such a rule means that the original program $\underline{e}$ is rewritten into another program $\underline{e}'$ assuming some hypothesis H in the context of a schema S and a variable environment Γ. The benefit of this rewriting is that $\underline{e}'$ is dead-code free. The rewriting is also safe in the sense that it preserves the semantics of

[3]SMIL is the standard language for expressing synchronized multimedia documents as found in e.g., MMS mobile phone messages, and more generally on the web [Hoschka, 1998].

$$\frac{S,\Gamma : \underline{e}_1 \longrightarrow ()}{S,\Gamma : \texttt{element}\{\underline{e}_1\}\{\underline{e}_2\} \longrightarrow ()} \qquad \frac{S,\Gamma : \underline{e}_i \longrightarrow \underline{e}'_i \qquad i = 1,2 \qquad \underline{e}'_1 \neq ()}{S,\Gamma : \texttt{element}\{\underline{e}_1\}\{\underline{e}_2\} \longrightarrow \texttt{element}\{\underline{e}'_1\}\{\underline{e}'_2\}}$$

$$\frac{S,\Gamma : \underline{e}_i \longrightarrow () \qquad i = 1,3}{S,\Gamma : \texttt{if } \underline{e}_1 \texttt{ then } \underline{e}_2 \texttt{ else } \underline{e}_3 \longrightarrow ()}$$

$$\frac{S,\Gamma : \underline{e}_1 \longrightarrow () \qquad S,\Gamma : \underline{e}_3 \longrightarrow \underline{e}'_3 \qquad \underline{e}'_3 \neq ()}{S,\Gamma : \texttt{if } \underline{e}_1 \texttt{ then } \underline{e}_2 \texttt{ else } \underline{e}_3 \longrightarrow \underline{e}'_3}$$

$$\frac{S,\Gamma : \underline{e}_i \longrightarrow \underline{e}'_i \qquad i = 1,2,3 \qquad \underline{e}'_1 \neq ()}{S,\Gamma : \texttt{if } \underline{e}_1 \texttt{ then } \underline{e}_2 \texttt{ else } \underline{e}_3 \longrightarrow \texttt{if } \underline{e}'_1 \texttt{ then } \underline{e}'_2 \texttt{ else } \underline{e}'_3}$$

$$\frac{S,\Gamma : \underline{e}_1 \longrightarrow ()}{S,\Gamma : \texttt{for } \underline{\$var} \texttt{ in } \underline{e}_1 \ (\texttt{where } \underline{e}_2)? \texttt{ return } \underline{e}_3 \longrightarrow ()}$$

$$\frac{S,\Gamma \cup (\underline{\$var}, \underline{e}_1) : \underline{e}_3 \longrightarrow ()}{S,\Gamma : \texttt{for } \underline{\$var} \texttt{ in } \underline{e}_1 \ (\texttt{where } \underline{e}_2)? \texttt{ return } \underline{e}_3 \longrightarrow ()}$$

$$\frac{S,\Gamma : \underline{e}_1 \longrightarrow \underline{e}'_1 \qquad \underline{e}'_1 \neq () \qquad S,\Gamma \cup (\underline{\$var}, \underline{e}'_1) : \underline{e}_2 \longrightarrow ()}{S,\Gamma : \texttt{for } \underline{\$var} \texttt{ in } \underline{e}_1 \texttt{ where } \underline{e}_2 \texttt{ return } \underline{e}_3 \longrightarrow ()}$$

Figure 4.15: XQuery Refactoring Rules.

the original program: executing the rewritten program yields the same result than executing the original program. The only difference is that $\underline{e}'$ is smaller than $\underline{e}$ in terms of code size, and thus potentially executes faster.

One of the most basic rules consists in replacing an inconsistent path by the empty node sequence, as follows:

$$\frac{\neg\text{satisfiable}(\underline{\text{path}}, S)}{S,\Gamma : \underline{\text{path}} \longrightarrow ()}$$

where the predicate satisfiable($\underline{\text{path}}, S$) in the hypothesis is the boolean test directly performed by the logical solver [Genevès et al., 2007b]. This rewriting is extended to other XQuery statements. Figure 4.15 details the rewriting principle for three main XQuery constructs, namely the instruction for generating elements, the "if" statement and the "for" loop. For instance, the first rule eliminates the instruction `element`$\{\underline{e}_1\}\{\underline{e}_2\}$ (rewrites it to the empty sequence) provided the expression $\underline{e}_1$ rewrites itself to the empty sequence. The third rule eliminates a whole `if` statement whenever both the `if` condition and the `else` clause rewrite to the empty sequence. Rules for other XQuery constructs follow the same principle and are similar.

4.8.5 Syntax Highlighting & Code Refactoring

The typical integrated development environment allows one to open an XQuery program and to associate with it a schema[4]. The code analysis process is

[4] A variety of schemas are actually supported including DTDs, XML Schemas and Relax NG definitions (see [Genevès et al., 2007b] for details).

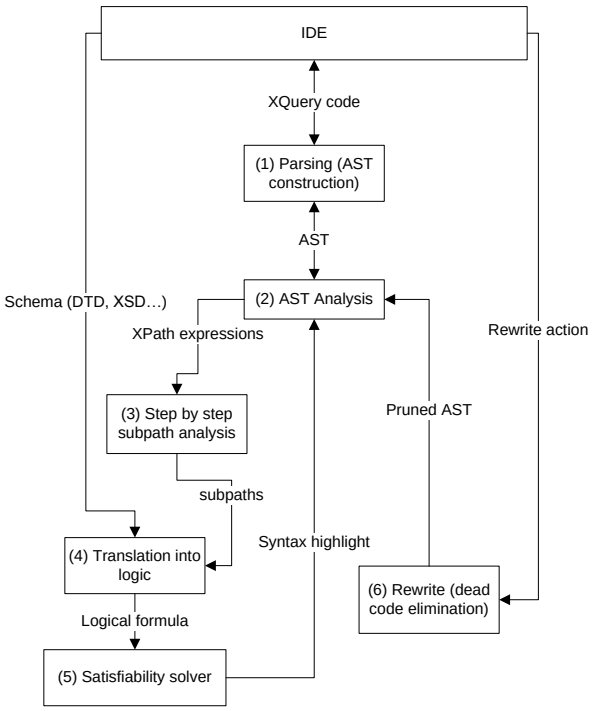


Figure 4.16: Code Analysis Diagram.

illustrated in Figure 4.16. First, the program is parsed to build an abstract syntax tree (step 1 in Figure 4.16). The abstract syntax tree (AST) analysis phase consists in extracting all the path expressions from the program and checking their satisfiability individually (steps 2 to 5). Then, in a second step, these paths are combined with the schema, and checked again for satisfiability (steps 2 to 5 again). This if for clearly distinguishing inconsistent paths (e.g. `child::a/child::b[parent::c]`) from inconsistent paths in the presence of the schema. Each kind of inconsistent path is marked differently in the AST. This makes it possible to inform the programmer, by underlining the empty path expressions in a different color depending on the origin of the inconsistency (self-contradiction or inconsistency in the presence of the given schema). More specifically, each path is considered as a sequence of basic navigation steps possibly with qualifiers. The first step is analyzed. Then each additional step is successively appended to this initial step and the resulting path is analyzed in turn (step 3). This makes it possible to identify precisely where the error has been introduced in the path. For instance, in the previous example, this step by step subpath analysis identifies the qualifier `parent::c` as causing the error.

Whenever an inconsistent path is found, a refactoring command is provided to the IDE user. When this command is triggered, the AST is pruned using the rules presented earlier (step 6), and the new XQuery program is provided

to the user.

4.9 Conclusion

We found very interesting that the practical restriction to trees and cycle-free formulas led to theoretical implications such as the collapsing of the least and greatest fixpoints. We were able to leverage this property to obtain an efficient implementation.

We also emphasize that the fact that the algorithm is implemented as a least fixpoint construction makes it possible to check for satisfiable formulas as soon as possible, often not requiring the full fixpoint to be computed, as opposed to algorithms based on greatest fixpoint computations that must eliminate all contradictions and therefore complete the computation of the fixpoint all the time.

The main result of this work is a sound and complete satisfiability-testing algorithm for a sub-logic of the alternation-free modal μ-calculus with converse for finite trees. The algorithm operates in time complexity $2^{O(n)}$ in the length n of a formula. It has been implemented and is available online.

As a direction for future work, we plan to study the extension of the logic with counting operators. We have started this investigation for restricted form of counting and interleaving [Bárcenas-Patiño, 2011], which we want to extend to the full logic. As another perspective, notice that it is possible to configure the solver such that, instead of computing one satisfying tree for a satisfiable formula, it computes a regular tree type representation of the set of all satisfying trees. Such a representation could be used in the setting of rich type systems for programming and query languages such as XQuery [Boag et al., 2006] and CDuce [Benzaken et al., 2003].

4.10 XPath and Regular Tree Languages

XPath [Clark & DeRose, 1999] is a powerful language for navigating in XML documents and selecting sets of nodes matching a predicate. In their simplest form, XPath expressions look like "directory navigation paths". For example, the XPath expression

```
/child::book/child::chapter/child::section
```

navigates from the root of a document (designated by the leading "/") through the top-level "book" node to its "chapter" child nodes and on to its child nodes named "section". The result of the evaluation of the entire expression is the set of all the "section" nodes that can be reached in this manner. The situation becomes more interesting when combined with XPath's capability of searching along "axes" other than "child". For instance, one may use the "preceding-sibling" axis for navigating backward through nodes of the same parent, or the "ancestor" axis for navigating upward recursively. Furthermore, at each step in the navigation the selected nodes can be filtered using qualifiers:

Boolean expression between brackets that can test the existence or absence of paths.

For the practical experiments, we consider an XPath fragment covering all major features of the XPath 1.0 recommendation [Clark & DeRose, 1999] with the exception of counting and comparisons between data values.

Figure 4.17 gives the syntax of XPath expressions. Figure 4.18 and Figure 4.19 give an interpretation of XPath expressions as functions between sets of focused trees.

$\mathcal{L}_{\text{XPath}} \ni e$		::=		XPath expression
			$/p$	absolute path
		|	p	relative path
		|	$e_1 \cup e_2$	union
		|	$e_1 \cap e_2$	intersection
Path	p	::=		path
			p_1/p_2	path composition
		|	$p[q]$	qualified path
		|	$\underline{\text{a}}::\sigma$	step with node test
		|	$\underline{\text{a}}::*$	step
Qualif	q	::=		qualifier
			q_1 and q_2	conjunction
		|	q_1 or q_2	disjunction
		|	not q	negation
		|	p	path
Axis	$\underline{a}$	::=		tree navigation axis
			child | self | parent	
		|	descendant | desc-or-self	
		|	ancestor | anc-or-self	
		|	foll-sibling | prec-sibling	
		|	following | preceding	

Figure 4.17: XPath Abstract Syntax.

4.10.1 XPath Embedding

We now explain how an XPath expression can be translated into an equivalent $\mathcal{L}_\mu$ formula that performs navigation in focused trees in binary style.

Logical Interpretation of Axes. The translation of navigational primitives, namely XPath axes, is formally specified in Figure 4.20. The translation function, noted "$A^{\rightarrow}[\![\underline{a}]\!]_\chi$", takes an XPath axis $\underline{a}$ as input, and returns its $\mathcal{L}_\mu$ translation, parameterized by the $\mathcal{L}_\mu$ formula χ given as parameter. This parameter represents the context in which the axis occurs and is needed for formula composition in order to translate path composition. More precisely, the formula $A^{\rightarrow}[\![\underline{a}]\!]_\chi$ holds for all nodes that can be accessed through the axis $\underline{a}$ from some node verifying χ.

Let us consider the path expression $A^{\rightarrow}[\![\text{child}]\!]_\chi$, translated as $\mu Z.\, \langle \overline{1} \rangle\, \chi \vee \langle \overline{2} \rangle\, Z$, which is satisfied by children of the context χ. These nodes consist of

$$
\begin{aligned}
\mathcal{S}_e[\![\cdot]\!]_\cdot &: \mathcal{L}_{\text{XPath}} \to 2^{\mathcal{F}} \to 2^{\mathcal{F}} \\
\mathcal{S}_e[\![/p]\!]_F &\stackrel{\text{def}}{=} \mathcal{S}_p[\![p]\!]_{\texttt{root}(F)} \\
\mathcal{S}_e[\![p]\!]_F &\stackrel{\text{def}}{=} \mathcal{S}_p[\![p]\!]_{\{(\sigma^{\mathbb{S}}[tl],c)\in F\}} \\
\mathcal{S}_e[\![e_1 \mid e_2]\!]_F &\stackrel{\text{def}}{=} \mathcal{S}_e[\![e_1]\!]_F \cup \mathcal{S}_e[\![e_2]\!]_F \\
\mathcal{S}_e[\![e_1 \cap e_2]\!]_F &\stackrel{\text{def}}{=} \mathcal{S}_e[\![e_1]\!]_F \cap \mathcal{S}_e[\![e_2]\!]_F
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{S}_p[\![\cdot]\!]_\cdot &: \underline{\text{Path}} \to 2^{\mathcal{F}} \to 2^{\mathcal{F}} \\
\mathcal{S}_p[\![p_1/p_2]\!]_F &\stackrel{\text{def}}{=} \left\{f' \mid f' \in \mathcal{S}_p[\![p_2]\!]_{(\mathcal{S}_p[\![p_1]\!]_F)}\right\} \\
\mathcal{S}_p[\![p[q]]\!]_F &\stackrel{\text{def}}{=} \{f \mid f \in \mathcal{S}_p[\![p]\!]_F \wedge \mathcal{S}_q[\![q]\!]_f\} \\
\mathcal{S}_p[\![\underline{\text{a}}::\sigma]\!]_F &\stackrel{\text{def}}{=} \left\{f \mid f \in \mathcal{S}_{\underline{a}}[\![\underline{\text{a}}]\!]_F \wedge \texttt{nm}(f) = \sigma\right\} \\
\mathcal{S}_p[\![\underline{\text{a}}::*]\!]_F &\stackrel{\text{def}}{=} \left\{f \mid f \in \mathcal{S}_{\underline{a}}[\![\underline{\text{a}}]\!]_F\right\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{S}_q[\![\cdot]\!]_\cdot &: \underline{\text{Qualif}} \to \mathcal{F} \to \{\text{true}, \text{false}\} \\
\mathcal{S}_q[\![q_1 \text{ and } q_2]\!]_f &\stackrel{\text{def}}{=} \mathcal{S}_q[\![q_1]\!]_f \wedge \mathcal{S}_q[\![q_2]\!]_f \\
\mathcal{S}_q[\![q_1 \text{ or } q_2]\!]_f &\stackrel{\text{def}}{=} \mathcal{S}_q[\![q_1]\!]_f \vee \mathcal{S}_q[\![q_2]\!]_f \\
\mathcal{S}_q[\![\text{not } q]\!]_f &\stackrel{\text{def}}{=} \neg\, \mathcal{S}_q[\![q]\!]_f \\
\mathcal{S}_q[\![p]\!]_f &\stackrel{\text{def}}{=} \mathcal{S}_p[\![p]\!]_{\{f\}} \neq \emptyset
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{S}_{\underline{a}}[\![\cdot]\!]_\cdot &: \underline{\text{Axis}} \to 2^{\mathcal{F}} \to 2^{\mathcal{F}} \\
\mathcal{S}_{\underline{a}}[\![\text{self}]\!]_F &\stackrel{\text{def}}{=} F \\
\mathcal{S}_{\underline{a}}[\![\text{child}]\!]_F &\stackrel{\text{def}}{=} \texttt{fchild}(F) \cup \mathcal{S}_{\underline{a}}[\![\text{foll-sibling}]\!]_{\texttt{fchild}(F)} \\
\mathcal{S}_{\underline{a}}[\![\text{foll-sibling}]\!]_F &\stackrel{\text{def}}{=} \texttt{nsibling}(F) \cup \mathcal{S}_{\underline{a}}[\![\text{foll-sibling}]\!]_{\texttt{nsibling}(F)} \\
\mathcal{S}_{\underline{a}}[\![\text{prec-sibling}]\!]_F &\stackrel{\text{def}}{=} \texttt{psibling}(F) \cup \mathcal{S}_{\underline{a}}[\![\text{prec-sibling}]\!]_{\texttt{psibling}(F)} \\
\mathcal{S}_{\underline{a}}[\![\text{parent}]\!]_F &\stackrel{\text{def}}{=} \texttt{parent}(F) \\
\mathcal{S}_{\underline{a}}[\![\text{descendant}]\!]_F &\stackrel{\text{def}}{=} \mathcal{S}_{\underline{a}}[\![\text{child}]\!]_F \cup \mathcal{S}_{\underline{a}}[\![\text{descendant}]\!]_{(\mathcal{S}_{\underline{a}}[\![\text{child}]\!]_F)} \\
\mathcal{S}_{\underline{a}}[\![\text{desc-or-self}]\!]_F &\stackrel{\text{def}}{=} F \cup \mathcal{S}_{\underline{a}}[\![\text{descendant}]\!]_F \\
\mathcal{S}_{\underline{a}}[\![\text{ancestor}]\!]_F &\stackrel{\text{def}}{=} \mathcal{S}_{\underline{a}}[\![\text{parent}]\!]_F \cup \mathcal{S}_{\underline{a}}[\![\text{ancestor}]\!]_{(\mathcal{S}_{\underline{a}}[\![\text{parent}]\!]_F)} \\
\mathcal{S}_{\underline{a}}[\![\text{anc-or-self}]\!]_F &\stackrel{\text{def}}{=} F \cup \mathcal{S}_{\underline{a}}[\![\text{ancestor}]\!]_F \\
\mathcal{S}_{\underline{a}}[\![\text{following}]\!]_F &\stackrel{\text{def}}{=} \mathcal{S}_{\underline{a}}[\![\text{desc-or-self}]\!]_{\left(\mathcal{S}_{\underline{a}}[\![\text{foll-sibling}]\!]_{(\mathcal{S}_{\underline{a}}[\![\text{anc-or-self}]\!]_F)}\right)} \\
\mathcal{S}_{\underline{a}}[\![\text{preceding}]\!]_F &\stackrel{\text{def}}{=} \mathcal{S}_{\underline{a}}[\![\text{desc-or-self}]\!]_{\left(\mathcal{S}_{\underline{a}}[\![\text{prec-sibling}]\!]_{(\mathcal{S}_{\underline{a}}[\![\text{anc-or-self}]\!]_F)}\right)}
\end{aligned}
$$

Figure 4.18: Interpretation of XPath Expressions as Functions Between Sets of Focused Trees.

$$
\begin{aligned}
\texttt{fchild}(F) &\stackrel{\text{def}}{=} \{f\langle 1\rangle \mid f \in F \wedge f\langle 1\rangle \text{ defined}\}\\
\texttt{nsibling}(F) &\stackrel{\text{def}}{=} \{f\langle 2\rangle \mid f \in F \wedge f\langle 2\rangle \text{ defined}\}\\
\texttt{psibling}(F) &\stackrel{\text{def}}{=} \{f\langle \overline{2}\rangle \mid f \in F \wedge f\langle \overline{2}\rangle \text{ defined}\}\\
\texttt{parent}(F) &\stackrel{\text{def}}{=} \{(\sigma^{\circ}[\texttt{rev\_a}(tl_l, t :: tl_r)], c)\\
&\qquad\qquad \mid (t, (tl_l, c[\sigma^{\circ}], tl_r)) \in F\}\\
\texttt{rev\_a}(\epsilon, tl_r) &\stackrel{\text{def}}{=} tl_r\\
\texttt{rev\_a}(t :: tl_l, tl_r) &\stackrel{\text{def}}{=} \texttt{rev\_a}(tl_l, t :: tl_r)\\
\texttt{root}(F) &\stackrel{\text{def}}{=} \{(\sigma^{\textcircled{S}}[tl], (tl, \mathit{Top}, tl)) \in F\}\\
&\qquad\qquad \cup\, \texttt{root}(\texttt{parent}(F))
\end{aligned}
$$

Figure 4.19: Auxiliary Functions for XPath Interpretation.

the first child and the remaining children. From the first child, the context must be reached immediately by going once upward via $\overline{1}$. From the remaining children, the context is reached by going upward (any number of times) via $\overline{2}$ and finally once via $\overline{1}$.

Logical Interpretation of Expressions. Figure 4.21 gives the translation of XPath expressions into $\mathcal{L}_\mu$. The translation function "$E^{\rightarrow}[\![e]\!]_\chi$" takes an XPath expression e and a $\mathcal{L}_\mu$ formula χ as input, and returns the corresponding $\mathcal{L}_\mu$ translation. The translation of a relative XPath expression marks the initial context with Ⓢ. The translation of an absolute XPath expression navigates to the root which is taken as the initial context.

Figure 4.22 illustrates the translation of the expression "child::a[child::b]". This expression selects all "a" child nodes of a given context which have at least one "b" child. The translated $\mathcal{L}_\mu$ formula holds for "a" nodes which are selected by the expression. The first part of the translated formula, φ, corresponds to the step "child::a" which selects candidates "a" nodes. The second part, ψ, navigates downward in the subtrees of these candidate nodes to verify that they have at least one immediate "b" child.

Note that without converse programs we would have been unable to differentiate selected nodes from nodes whose existence is tested: we must state properties on both the ancestors and the descendants of the selected node. The fact that the $\mathcal{L}_\mu$ logic is equipped with both forward and converse programs is important for supporting XPath[5]. Logics without converse programs may only be used for solving XPath emptiness but cannot be used for solving other decision problems such as containment efficiently.

[5]One may ask whether it is possible to eliminate upward navigation at the XPath level but it is well known that such XPath rewriting techniques cause exponential blow-ups of expression sizes [Olteanu et al., 2002].

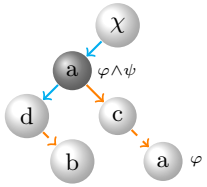
$$
\begin{aligned}
A^{\rightarrow}[\![\cdot]\!]_{\cdot} &: \underline{\text{Axis}} \to \mathcal{L}_\mu \to \mathcal{L}_\mu \\
A^{\rightarrow}[\![\text{self}]\!]_\chi &\stackrel{\text{def}}{=} \chi \\
A^{\rightarrow}[\![\text{child}]\!]_\chi &\stackrel{\text{def}}{=} \mu Z.\, \langle \overline{1} \rangle\, \chi \vee \langle \overline{2} \rangle\, Z \\
A^{\rightarrow}[\![\text{foll-sibling}]\!]_\chi &\stackrel{\text{def}}{=} \mu Z.\, \langle \overline{2} \rangle\, \chi \vee \langle \overline{2} \rangle\, Z \\
A^{\rightarrow}[\![\text{prec-sibling}]\!]_\chi &\stackrel{\text{def}}{=} \mu Z.\, \langle 2 \rangle\, \chi \vee \langle 2 \rangle\, Z \\
A^{\rightarrow}[\![\text{parent}]\!]_\chi &\stackrel{\text{def}}{=} \langle 1 \rangle\, \mu Z.\chi \vee \langle 2 \rangle\, Z \\
A^{\rightarrow}[\![\text{descendant}]\!]_\chi &\stackrel{\text{def}}{=} \mu Z.\, \langle \overline{1} \rangle\, (\chi \vee Z) \vee \langle \overline{2} \rangle\, Z \\
A^{\rightarrow}[\![\text{desc-or-self}]\!]_\chi &\stackrel{\text{def}}{=} \mu Z.\chi \vee \mu Y.\, \langle \overline{1} \rangle\, (Y \vee Z) \vee \langle \overline{2} \rangle\, Y \\
A^{\rightarrow}[\![\text{ancestor}]\!]_\chi &\stackrel{\text{def}}{=} \langle 1 \rangle\, \mu Z.\chi \vee \langle 1 \rangle\, Z \vee \langle 2 \rangle\, Z \\
A^{\rightarrow}[\![\text{anc-or-self}]\!]_\chi &\stackrel{\text{def}}{=} \mu Z.\chi \vee \langle 1 \rangle\, \mu Y.Z \vee \langle 2 \rangle\, Y \\
A^{\rightarrow}[\![\text{following}]\!]_\chi &\stackrel{\text{def}}{=} A^{\rightarrow}[\![\text{desc-or-self}]\!]_{\eta_1} \\
A^{\rightarrow}[\![\text{preceding}]\!]_\chi &\stackrel{\text{def}}{=} A^{\rightarrow}[\![\text{desc-or-self}]\!]_{\eta_2} \\
\eta_1 &\stackrel{\text{def}}{=} A^{\rightarrow}[\![\text{foll-sibling}]\!]_{A^{\rightarrow}[\![\text{anc-or-self}]\!]_\chi} \\
\eta_2 &\stackrel{\text{def}}{=} A^{\rightarrow}[\![\text{prec-sibling}]\!]_{A^{\rightarrow}[\![\text{anc-or-self}]\!]_\chi}
\end{aligned}
$$

Figure 4.20: Translation of XPath Axes.

$$
\begin{aligned}
E^{\rightarrow}[\![\cdot]\!]_{\cdot} &: \mathcal{L}_{\text{XPath}} \to \mathcal{L}_\mu \to \mathcal{L}_\mu \\
E^{\rightarrow}[\![/p]\!]_\chi &\stackrel{\text{def}}{=} P^{\rightarrow}[\![p]\!]_{\left(\left(\mu Z.\neg\langle \overline{1} \rangle \top \vee \langle \overline{2} \rangle Z\right) \wedge \left(\mu Y.\chi \wedge \textcircled{\scriptsize S} \vee \langle 1 \rangle Y \vee \langle 2 \rangle Y\right)\right)} \\
E^{\rightarrow}[\![p]\!]_\chi &\stackrel{\text{def}}{=} P^{\rightarrow}[\![p]\!]_{(\chi \wedge \textcircled{\scriptsize S})} \\
E^{\rightarrow}[\![e_1 \mid e_2]\!]_\chi &\stackrel{\text{def}}{=} E^{\rightarrow}[\![e_1]\!]_\chi \vee E^{\rightarrow}[\![e_2]\!]_\chi \\
E^{\rightarrow}[\![e_1 \cap e_2]\!]_\chi &\stackrel{\text{def}}{=} E^{\rightarrow}[\![e_1]\!]_\chi \wedge E^{\rightarrow}[\![e_2]\!]_\chi
\end{aligned}
$$

$$
\begin{aligned}
P^{\rightarrow}[\![\cdot]\!]_{\cdot} &: \underline{\text{Path}} \to \mathcal{L}_\mu \to \mathcal{L}_\mu \\
P^{\rightarrow}[\![p_1/p_2]\!]_\chi &\stackrel{\text{def}}{=} P^{\rightarrow}[\![p_2]\!]_{(P^{\rightarrow}[\![p_1]\!]_\chi)} \\
P^{\rightarrow}[\![p[q]]\!]_\chi &\stackrel{\text{def}}{=} P^{\rightarrow}[\![p]\!]_\chi \wedge Q^{\leftarrow}[\![q]\!]_\top \\
P^{\rightarrow}[\![\underline{\text{a}}::\sigma]\!]_\chi &\stackrel{\text{def}}{=} \sigma \wedge A^{\rightarrow}[\![\underline{a}]\!]_\chi \\
P^{\rightarrow}[\![\underline{\text{a}}::*]\!]_\chi &\stackrel{\text{def}}{=} A^{\rightarrow}[\![\underline{a}]\!]_\chi
\end{aligned}
$$

Figure 4.21: Translation of Expressions and Paths.



Translated Expression: child::a[child::b]

$$\underbrace{a \wedge (\mu X.\, \langle \overline{1} \rangle\, (\chi \wedge \textcircled{S}) \vee \langle \overline{2} \rangle\, X)}_{\varphi} \wedge \underbrace{\langle 1 \rangle\, \mu Y.b \vee \langle 2 \rangle\, Y}_{\psi}$$

Figure 4.22: XPath Translation Example.

XPath composition construct p_1/p_2 translates into formula composition in $\mathcal{L}_\mu$, such that the resulting formula holds for all nodes accessed through p_2 from those nodes accessed through p_1 from χ. The translation of the branching construct $p[q]$ significantly differs. The resulting formula must hold for all nodes that can be accessed through p and from which q holds. To preserve semantics, the translation of $p[q]$ stops the "selecting navigation" to those nodes reached by p, then filters them depending on whether q holds or not. We express this by introducing a dual formal translation function for XPath qualifiers, noted $Q^{\leftarrow}[\![q]\!].$ and defined in Figure 4.23, that performs "filtering" instead of navigation. Specifically, $P^{\rightarrow}[\![\cdot]\!].$ can be seen as the "navigational" translating function: the translated formula holds for target nodes of the given path. On the opposite, $Q^{\leftarrow}[\![\cdot]\!].$ can be seen as the "filtering" translating function: it states the existence of a path without moving to its result. The translated formula $Q^{\leftarrow}[\![q]\!]_\chi$ (respectively $P^{\leftarrow}[\![p]\!]_\chi$) holds for nodes from which there exists a qualifier q (respectively a path p) leading to a node verifying χ.

XPath translation is based on these two translating "modes", the first one being used for paths and the second one for qualifiers. Whenever the "filtering" mode is entered, it will never be left.

The translation of paths inside qualifiers is also given in Figure 4.23. It uses the translation for axes and is based on XPath symmetry: symmetric(a) denotes the symmetric XPath axis corresponding to the axis a (for instance symmetric(child) = parent).

We may now state that our translation is correct, by relating the interpretation of an XPath formula applied to some set of trees to the interpretation of its translation, by stating that the translation of a formula is cycle-free, and by giving a bound in the size of this translation.

We restrict the sets of trees to which an XPath formula may be applied to those that may be denoted by an $\mathcal{L}_\mu$ formula. This restriction will be justified in Section 4.10.2 where we show that every regular tree language may be translated to an $\mathcal{L}_\mu$ formula.

Proposition 4.10.1 (Translation Correctness)**.** *The following hold for an*

$$
\begin{aligned}
Q^{\leftarrow}[\![\cdot]\!]_{\cdot} &: \underline{\text{Qualif}} \to \mathcal{L}_\mu \to \mathcal{L}_\mu \\
Q^{\leftarrow}[\![q_1 \text{ and } q_2]\!]_\chi &\stackrel{\text{def}}{=} Q^{\leftarrow}[\![q_1]\!]_\chi \wedge Q^{\leftarrow}[\![q_2]\!]_\chi \\
Q^{\leftarrow}[\![q_1 \text{ or } q_2]\!]_\chi &\stackrel{\text{def}}{=} Q^{\leftarrow}[\![q_1]\!]_\chi \vee Q^{\leftarrow}[\![q_2]\!]_\chi \\
Q^{\leftarrow}[\![\text{not } q]\!]_\chi &\stackrel{\text{def}}{=} \neg\, Q^{\leftarrow}[\![q]\!]_\chi \\
Q^{\leftarrow}[\![p]\!]_\chi &\stackrel{\text{def}}{=} P^{\leftarrow}[\![p]\!]_\chi
\end{aligned}
$$

$$
\begin{aligned}
P^{\leftarrow}[\![\cdot]\!]_{\cdot} &: \underline{\text{Path}} \to \mathcal{L}_\mu \to \mathcal{L}_\mu \\
P^{\leftarrow}[\![p_1/p_2]\!]_\chi &\stackrel{\text{def}}{=} P^{\leftarrow}[\![p_1]\!]_{(P^{\leftarrow}[\![p_2]\!]_\chi)} \\
P^{\leftarrow}[\![p[q]]\!]_\chi &\stackrel{\text{def}}{=} P^{\leftarrow}[\![p]\!]_{(\chi \wedge Q^{\leftarrow}[\![q]\!]_\top)} \\
P^{\leftarrow}[\![\underline{\text{a}}::\sigma]\!]_\chi &\stackrel{\text{def}}{=} A^{\leftarrow}[\![\underline{a}]\!]_{(\chi \wedge \sigma)} \\
P^{\leftarrow}[\![\underline{\text{a}}::*]\!]_\chi &\stackrel{\text{def}}{=} A^{\leftarrow}[\![\underline{a}]\!]_\chi
\end{aligned}
$$

$$
\begin{aligned}
A^{\leftarrow}[\![\cdot]\!]_{\cdot} &: \underline{\text{Axis}} \to \mathcal{L}_\mu \to \mathcal{L}_\mu \\
A^{\leftarrow}[\![\underline{a}]\!]_\chi &\stackrel{\text{def}}{=} A^{\rightarrow}[\![\underline{\text{symmetric}}(\underline{a})]\!]_\chi
\end{aligned}
$$

Figure 4.23: Translation of Qualifiers.

XPath expression e and a $\mathcal{L}_\mu$ formula φ denoting a set of focused trees, with $\psi = E^{\rightarrow}[\![e]\!]_\varphi$:

1. $[\![\psi]\!]_\emptyset = \mathcal{S}_e[\![e]\!]_{[\![\varphi]\!]_\emptyset}$
2. *ψ is cycle-free*
3. *the size of ψ is linear in the size of e and φ*

Proof: The proof uses a structural induction that "peels off" the compositional layers of each set of rules over focused trees. The cycle-free part follows from the fact that translated fixpoint formulas are closed and there is no nesting of modalities with converse programs between a fixpoint variable and its binder. Each XPath navigation step is cycle-free, and their composition yields a proper nesting of fixpoint formulas which is also cycle-free. Figure 4.24 illustrates this on an typical example. Finally, formal translations do not duplicate any subformula of arbitrary length. □

4.10.2 Embedding Regular Tree Languages

Several formalisms exist for describing types of XML documents (e.g. DTD, XML Schema, Relax NG). In our case we embed regular tree types into $\mathcal{L}_\mu$. Regular tree types gather most of the schemas occuring in practice [Murata

Translation of following-sibling::a/preceding-sibling::b
into $\mathcal{L}_\mu$: $b \wedge [\mu Y.\, \langle 2\rangle\, (a \wedge (\mu Z.\, \langle \overline{2}\rangle\, \text{Ⓢ} \vee \langle \overline{2}\rangle\, Z)\,) \vee \langle 2\rangle\, Y]$

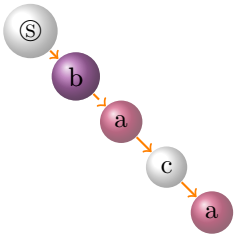


Figure 4.24: Example of Back and Forth – Yet Cycle-Free – XPath Navigation.

et al., 2005][6]. We rely on a straightforward isomorphism between unranked regular tree types and binary regular tree types [Hosoya et al., 2005a]. Assuming a countably infinite set of type variables ranged over by X, binary regular tree type expressions are defined as follows:

$\mathcal{L}_{\text{bt}} \ni T$	::=		tree type expression
		$\emptyset$	empty set
	$\mid$	ϵ	leaf
	$\mid$	$T_1 \mid T_2$	union
	$\mid$	$\sigma(X_1, X_2)$	label
	$\mid$	let $\overline{X_i.T_i}$ in T	binder

We refer the reader to [Hosoya et al., 2005a] for the denotational semantics of regular tree languages, and directly introduce their translation into $\mathcal{L}_\mu$:

$$
\begin{aligned}
[\![\cdot]\!] &: \mathcal{L}_{\text{bt}} \to \mathcal{L}_\mu \\
[\![T]\!] &\stackrel{\text{def}}{=} p \wedge \neg p \quad \text{for} \quad T = \emptyset, \epsilon \\
[\![T_1 \mid T_2]\!] &\stackrel{\text{def}}{=} [\![T_1]\!] \vee [\![T_2]\!] \\
[\![\sigma(X_1, X_2)]\!] &\stackrel{\text{def}}{=} \sigma \wedge \underline{\text{succ}}_1(X_1) \wedge \underline{\text{succ}}_2(X_2) \\
[\![\text{let } \overline{X_i.T_i} \text{ in } T]\!] &\stackrel{\text{def}}{=} \mu\overline{X_i.[\![T_i]\!]} \text{ in } [\![T]\!]
\end{aligned}
$$

where we use the formula $p \wedge \neg p$ as "false", and the function $\underline{\text{succ}}_\cdot(\cdot)$ takes care of setting the type frontier:

$$
\underline{\text{succ}}_\alpha(X) \quad = \quad \begin{cases} \neg\langle\alpha\rangle\top & \underline{\text{if } X \text{ is bound to } \epsilon} \\ \neg\langle\alpha\rangle\top \vee \langle\alpha\rangle X & \underline{\text{if nullable}}(X) \\ \langle\alpha\rangle X & \underline{\text{if not nullable}}(X) \end{cases}
$$

according to the predicate $\underline{\text{nullable}}(X)$ which indicates whether the type $T \neq \epsilon$ bound to X contains the empty tree. For example, Figure 4.27 gives the

[6] Notice, however, that we do not consider counting nor interleaving features that can be found in e.g. XML Schemas. These features are beyond the scope of this document: see [Bárcenas-Patiño, 2011] for a preliminary work on how to integrate counting constraints in such a logic.

```
<!ELEMENT article (meta, (text | redirect))>
<!ELEMENT meta (title, status?, interwiki*, history?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT interwiki (#PCDATA)>
<!ELEMENT status (#PCDATA)>
<!ELEMENT history (edit)+>
<!ELEMENT edit (status?, interwiki*, (text | redirect)?)>
<!ELEMENT redirect EMPTY>
<!ELEMENT text (#PCDATA)>
```

Figure 4.25: A Fragment of the DTD of the Wikipedia Encyclopedia.

```
$9 ->EPSILON
        | text($Epsilon, $Epsilon)
        | redirect($Epsilon, $Epsilon)
        | interwiki($Epsilon, $9)
$6 ->EPSILON
        | text($Epsilon, $Epsilon)
        | redirect($Epsilon, $Epsilon)
        | interwiki($Epsilon, $9)
        | status($Epsilon, $9)
$5 ->edit($6, $Epsilon)
        | edit($6, $5)
$14 ->EPSILON
        | history($5, $Epsilon)
        | interwiki($Epsilon, $14)
$4 ->EPSILON
        | history($5, $Epsilon)
        | interwiki($Epsilon, $14)
        | status($Epsilon, $14)
$2 ->title($Epsilon, $4)
$17 ->text($Epsilon, $Epsilon)
        | redirect($Epsilon, $Epsilon)
$1 ->meta($2, $17)
$article ->article($1, $Epsilon)
Start Symbol is $article
9 type variables.
9 terminals.
```

Figure 4.26: The Binary Encoding of the DTD of Figure 4.25.

translation of a DTD fragment of the Wikipedia encyclopedia [Voss, 2007] shown on Figure 4.25. The intermediate binary tree type encoding of the DTD is shown on Figure 4.26.

Note that the translation of a regular tree type uses only downward modalities since it describes the allowed subtrees at a given context. No additional restriction is imposed on the context from which the type definition starts. In

```
(let_mu
  X2=((((text & ~(<1>T)) & ~(<2>T)) | ((redirect & ~(<1>T)) & ~(<2>T)))
     | ((interwiki & ~(<1>T)) & (~(<2>T) | <2>X2))),
  X3=(((((text & ~(<1>T)) & ~(<2>T)) | ((redirect & ~(<1>T)) & ~(<2>T)))
     | ((interwiki & ~(<1>T)) & (~(<2>T) | <2>X2)))
     | ((status & ~(<1>T)) & (~(<2>T) | <2>X2))),
  X4=(((edit & (~(<1>T) | <1>X3)) & ~(<2>T)) |
     ((edit & (~(<1>T) | <1>X3)) & <2>X4)),
  X5=(((history & <1>X4) & ~(<2>T)) |
     ((interwiki & ~(<1>T)) & (~(<2>T) | <2>X5))),
  X6=((((history & <1>X4) & ~(<2>T)) | ((interwiki & ~(<1>T)) & (~(<2>T)
     | <2>X5))) | ((status & ~(<1>T)) & (~(<2>T) | <2>X5))),
  X7=((title & ~(<1>T)) & (~(<2>T) | <2>X6)),
  X8=(((text & ~(<1>T)) & ~(<2>T)) | ((redirect & ~(<1>T)) & ~(<2>T))),
  X9=((meta & <1>X7) & <2>X8),
  X10=((article & <1>X9) & ~(<2>T))
in
  X10)
```

Figure 4.27: The $\mathcal{L}_\mu$ Formula for the DTD of Figure 4.25.

particular, navigation is allowed in the upward direction so that we can support type constraints for which we have only partial knowledge in a given direction. However, when we know the position of the root, conditions similar to those of absolute paths are added in the form of additional formulas describing the position that need to be satisfied. This is particularly useful when a regular type is used by an XPath expression that starts its navigation at the root ($/p$) since the path will not go above the root of the type (by adding the restriction $\mu Z.\neg \langle \overline{1} \rangle \top \vee \langle \overline{2} \rangle Z$).

On the other hand, if the type is compared with another type (typically to check inclusion of the result of an XPath expression in this type), then there is no restriction as to where the root of the type is (our translation does not impose the chosen node to be at the root). This is particularly useful since an XPath expression usually returns a set of nodes deep in the tree which we may compare to this partially defined type.

Five

Impact of XML Schema Evolution

Abstract

We consider the problem of XML Schema evolution. In the ever-changing context of the web, XML schemas continuously change in order to cope with the natural evolution of entities they describe. Schema changes have important consequences. First, existing documents valid with respect to the original schema are no longer guaranteed to fulfill the constraints described by the evolved schema. Second, the evolution also impacts programs manipulating documents whose structure is described by the original schema.

We propose a unifying framework for determining the effects of XML Schema evolution both on the validity of documents and on queries. The system is very powerful in analyzing various scenarios in which forward/backward compatibility of schemas is broken, and in which the result of a query may not be anymore what was expected. Specifically, the system offers a predicate language which allows one to formulate properties related to schema evolution. The system then relies on exact reasoning techniques to perform a fine-grained analysis. This yields either a formal proof of the property or a counter-example that can be used for debugging purposes. The system has been fully implemented and tested with real-world use cases, in particular with the main standard document formats used on the web, as defined by W3C. The system identifies precisely compatibility relations between document formats. In case these relations do not hold, the system can identify queries that must be reformulated in order to produce the expected results across successive schema versions.

5.1 Introduction

XML is now commonplace on the web and in many information systems where it is used for representing all kinds of information resources, ranging from simple text documents such as RSS or Atom feeds to highly structured databases. In these dynamic environments, not only data are changing steadily but their schemas also get modified to cope with the evolution of the real world entities they describe.

Schema changes raise the issue of data consistency. Existing documents and data that were valid with a certain version of a schema may become invalid on a new version of the schema (forward incompatibility). Conversely, new

documents created with the latest version of a schema may be invalid on some previous versions (backward incompatibility). In particular, there are two ways commonly used in the design of schemas. One consists in under constraining the schema in the earlier versions when the design is not completely stable and then constraining it in future versions progressively. The other way is more conservative and consists in constraining the schema first and then relaxing the constraints progressively. If we leave aside new elements and attributes introduced between two successive versions of a schema, this is particularly true for new combinations of elements (content models) added or restricted through regular expressions in W3C Document formats recommendations (see Section 5.5).

In addition, schemas may be written in different languages, such as DTD, XML Schema, or Relax-NG, to name only the most popular ones. And it is common practice to describe the same structure, or new versions of a structure, in different schema languages. Document formats developed by W3C provide a variety of examples: XHTML 1.0 has both DTDs and XML Schemas, while XHTML 2.0 has a Relax-NG definition; the schema for SVG Tiny 1.1 is a DTD, while version 1.2 is written in Relax-NG; MathML 1.01 has a DTD, MathML 2.0 has both a DTD and an XML Schema, and MathML 3.0 is developed with a Relax-NG schema and also published with a DTD and an XML Schema. An issue then is to make sure that schemas written in different languages are equivalent, i.e. they describe the same structure, possibly with some differences due to the expressivity of the language [Murata et al., 2005]. Another issue is to clearly identify the differences between two versions of the same schema expressed in different languages. Moreover, the issues of forward and backward compatibility of instances obviously remain when schema languages change from a version to another.

Validation, and then compatibility, is not the only purpose of a schema. Validation is usually the first step for safe processing of documents and data. It makes sure that documents and data are structured as expected and can then be processed safely. The next step is to actually access and select the various parts to be handled in each phase of an application. For this, query languages play a key role. As an example, when transforming a document with XSL, XPath queries are paramount to locate in the original document the data to be produced in the transformed document.

Queries are affected by schema evolutions. The structures they return may change depending on the version of the schema used by a document. When changing schema, a query may return nothing, or something different from what was expected, and obviously further processing based on this query is at risk.

These observations highlight the need for evaluating precisely and safely the impact of schema evolutions on existing and future instances of documents and data. They also show that it is important for software engineers to precisely know what parts of a processing chain have to be updated when schemas change. In this chapter we focus on the XPath query language which is used in

many situations while processing XML documents and data. The XSL transformation language was already mentioned, but XPath is also present in XLink and XQuery for instance.

A part of this work concerning the impact of schema changes on XPath queries was presented at the ACM International Conference on Functional Programming (ICFP), 2009, [Genevès et al., 2009]. The present article aims at covering the more general issue of schema evolution by taking into account the impact on the validity of documents as well. In particular, we identify criteria for the evolution of standard XML Schemas. We present a framework for checking these criteria with the schemas specifying the main standard documents formats used on the web, as defined by W3C (see Section 5.5).

5.1.1 Outline

We first introduce the framework from a high-level perspective in Section 5.2: we describe how the whole system is assembled, and which XML schemas and queries are supported. In Section 5.3, we provide a more in-depth understanding of the underlying logic on which the system is built; in particular we explain how XML constructs are mapped to this logical representation. Based on this logical encodings, Section 5.4 introduces a predicate language specifically designed for assessing the impact of schema evolutions. The following sections respectively focus on applying the framework for studying the impact of schema evolutions on the validity of documents (Section 5.5) and on queries (Section 5.6). The full implementation of the system is presented in Section 5.7. Finally, we discuss related work in Section 5.8 before concluding in Section 5.9.

5.2 Analysis Framework

The main contribution of this chapter is a unifying framework that allows the automatic verification of properties related to XML schema evolution and its impact on the validity of documents and on queries. In particular, it offers the possibility of checking fine-grained properties of the behavior of queries with respect to successive versions of a given schema. The system can be used for checking relations between schemas and whether schema evolutions require a particular query to be updated. Whenever schema evolutions may induce query malfunctions, the system is able to generate annotated XML documents that exemplify bugs, with the goal of helping the programmer to understand and properly overcome undesired effects of schema evolutions.

The system relies on a predicate language (presented in Section 5.4) specifically designed for studying schema and query compatibility issues when schemas evolve. Specifically, predicates allow characterizing in a precise manner nodes subject to evolution. For instance, predicates allow to distinguish new nodes selected by the query after a schema change from new nodes that appear in the modified schema. Predicates also allow to describe nodes that appear in new regions of a schema compared to its original version, or even in a new context described by a particular XPath expression. Predicates, together with

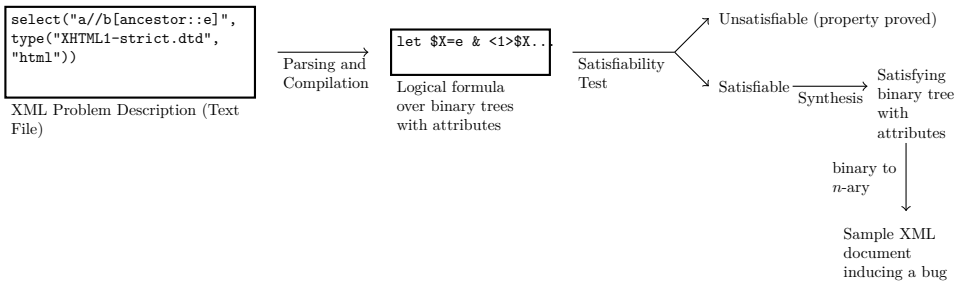


Figure 5.1: Framework Overview.

the composition language provided in the system allow to express and analyze complex settings.

The system has been fully implemented [Genevès & Layaïda, 2006a] and is outlined in Figure 5.1. It is composed of a parser for reading the text file description of the problem (which in turn uses specific parsers for schemas, queries, logical formulas, and predicates), compilers for translating schemas and queries into their logical representations, a solver for checking satisfiability of logical formulas, and a counter example XML tree generator (described in [Genevès et al., 2008]).

We first introduce the data model we consider for XML documents, schemas and queries.

5.2.1 XML Trees with Attributes

An XML document is considered as a finite tree of unbounded depth and arity, with two kinds of nodes respectively named elements and attributes. In such a tree, an element may have any number of children elements, and may carry zero, one or more attributes. Attributes are leaves. Elements are ordered whereas attributes are not, as illustrated on Figure 5.4. In this chapter, we focus on the nested structure of elements and attributes, and ignore XML data values.

5.2.2 Type Constraints

Our tree type expressions capture most of the schemas in use today either written using DTD, XML Schema, Relax NG, etc. Users may thus define constraints over XML documents with the language of their choice, and, more importantly, they may refer to most existing schemas for use with the system. Instead of having one parser/compiler per schema language, we rely on a common intermediate language in which all these languages are compiled. For the intermediate language we consider the standard class of regular tree grammars, commonly found in the literature [Hosoya et al., 2005b], to which we have added the support of constraints over XML attributes (whose efficiency is further discussed in section 5.3.3). In terms of expressive power, regular tree grammars support constraints over trees which are more expressive than local

tree grammars (DTDs) and single-type tree grammars (XML schemas), capturing exactly the class of Relax NG schemas, and, more fundamentally finite tree automata (see [Murata et al., 2005] for a formal characterization of the respective expressive power of these languages). In practice, we have implemented parsers that produce this intermediate representation from a given DTD, XML Schema, or Relax NG schema. We have implemented one compiler from this representation into the logic. An advantage of this approach is that it is extensible: it is easy to know the supported features since (1) the intermediate language is well-characterized and made explicit, and (2) extending the system with new schema languages is easy since one does not need to implement new compilers into the logic (and prove soundness, completeness and polynomial-time translation), but rather simply express the new considered constraints in the intermediate language.

Specifically, our unifying internal representation for tree grammars is made of regular tree type expressions, extended with constraints over attributes. Assuming a set of variables ranged over by x, we define a tree type expression as follows:

$\mathcal{T}$	::=		tree type expression
		$\emptyset$	empty set
	|	$()$	empty sequence
	|	$\mathcal{T} \mid \mathcal{T}$	disjunction
	|	$\mathcal{T}, \mathcal{T}$	concatenation
	|	$l(\underline{a})\,[\mathcal{T}]$	element definition
	|	x	variable
	|	let $\overline{x_i.\mathcal{T}_i}$ in $\mathcal{T}$	binder

The `let` construct allows binding one or more variables to associated formulas. Since several variables can be bound at a time, the notation $\overline{x = \mathcal{T}}$ is used for denoting a vector of variable bindings (possibly with mutual recursion).

We impose a usual restriction on the recursive use of variables: we allow unguarded (i.e. not enclosed by a label) recursive uses of variables, but restrict them to tail positions[1]. With that restriction, tree types expressions define regular tree languages. In addition, an element definition may involve simple attribute expressions that describe which attributes the defined element may (or may not) carry:

$\underline{a}$	::=		attribute expression
		$()$	empty list
	|	$\underline{list} \mid \underline{a}$	disjunction
$\underline{list}$	::=		attribute list
		$\underline{list}, \underline{list}$	commutative concatenation
	|	$l?$	optional attribute
	|	l	required attribute
	|	$\neg l$	prohibited attribute

[1] For instance, "let $x.l(\underline{a})\,[\mathcal{T}], x \mid ()$ in x" is allowed.

We use the usual semantics of regular tree types found in [Hosoya et al., 2005b] and [Genevès et al., 2008].

5.2.3 Queries

The set of XPath expressions we consider is given by the syntax shown on Figure 5.2. The semantics of XPath expressions is described in [Clark & DeRose, 1999], and more formally in [Wadler, 2000]. We observed that, in practice, many XPath expressions contain syntactic sugars that can also fit into this fragment. Figure 5.3 presents how our XPath parser rewrites some commonly found XPath patterns into the fragment of Figure 5.2, where the notation $(\underline{a}::\underline{nt})^k$ stands for the composition of k successive path steps of the same form: $\underbrace{\underline{a}::\underline{nt}/.../\underline{a}::\underline{nt}}_{k \text{ steps}}$.

$\underline{query}$	::=		
		$/\underline{path}$	absolute path
	|	$\underline{path}$	relative path
	|	$\underline{query} \mid \underline{query}$	union
	|	$\underline{query} \cap \underline{query}$	intersection
$\underline{path}$	::=		
		$\underline{path}/\underline{path}$	path composition
	|	$\underline{path}[\underline{qualifier}]$	qualified path
	|	$\underline{a}::\underline{nt}$	step
$\underline{qualifier}$	::=		
		$\underline{qualifier}$ and $\underline{qualifier}$	conjunction
	|	$\underline{qualifier}$ or $\underline{qualifier}$	disjunction
	|	not($\underline{qualifier}$)	negation
	|	$\underline{path}$	path
	|	$\underline{path}/@\underline{nt}$	attribute path
	|	$@\underline{nt}$	attribute step
$\underline{nt}$	::=		node test
		σ	node label
	|	$*$	any node label
$\underline{a}$	::=		tree navigation axis
		self | child | parent	
	|	descendant | ancestor	
	|	descendant-or-self	
	|	ancestor-or-self	
	|	following-sibling	
	|	preceding-sibling	
	|	following | preceding	

Figure 5.2: XPath Expressions.

The next Section presents the logic underlying the predicate language.

$$
\begin{aligned}
\underline{\text{nt}}[\text{position}() = 1] &\rightsquigarrow \underline{\text{nt}}[\text{not}(\text{preceding-sibling}::\underline{\text{nt}})] \\
\underline{\text{nt}}[\text{position}() = \text{last}()] &\rightsquigarrow \underline{\text{nt}}[\text{not}(\text{following-sibling}::\underline{\text{nt}})] \\
\underline{\text{nt}}[\text{position}() = \underbrace{k}_{k>1}] &\rightsquigarrow \underline{\text{nt}}[(\text{preceding-sibling}::\underline{\text{nt}})^{k-1}] \\
\text{count}(\underline{path}) = 0 &\rightsquigarrow \text{not}(\underline{path}) \\
\text{count}(\underline{path}) > 0 &\rightsquigarrow \underline{path} \\
\text{count}(\underline{\text{nt}}) > \underbrace{k}_{k>0} &\rightsquigarrow \underline{\text{nt}}/(\text{following-sibling}::\underline{\text{nt}})^{k}
\end{aligned}
$$

$$
\begin{aligned}
&\text{preceding-sibling}::*[\text{position}() = \text{last}() \text{ and } \underline{qualifier}] \\
&\qquad \rightsquigarrow \text{preceding-sibling}::*[\text{not}(\text{preceding-sibling}::*) \text{ and } \underline{qualifier}]
\end{aligned}
$$

Figure 5.3: Syntactic Sugars and their Rewritings.

5.3 Logical Setting

It is well-known that there exist bijective encodings between unranked trees (trees of unbounded arity) and binary trees [Thomas, 1990]. Owing to these encodings binary trees may be used instead of unranked trees without loss of generality. In the sequel, we rely on a simple "first-child & next-sibling" encoding of unranked trees. In this encoding, the first child of an element node is preserved in the binary tree representation, whereas siblings of this node are appended as right successors in the binary representation. Attributes are left unchanged by this encoding. For instance, Figure 5.5 presents how the sample tree of Figure 5.4 is mapped.

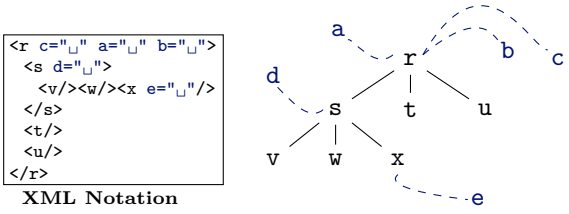


Figure 5.4: Sample XML Tree with Attributes.

The logic we introduce below, used as the core of our framework, operates on such binary trees with attributes.

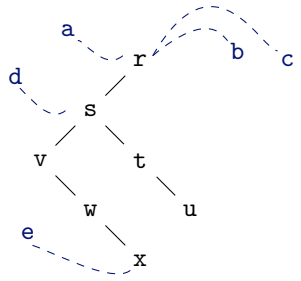


Figure 5.5: Binary Encoding of Tree of Figure 5.4.

5.3.1 Logical Formulas

The concrete syntax of logical formulas is shown on Figure 5.6, where the meta-syntax $\langle X \rangle^{\oplus}$ means one or more occurences of X separated by commas. The user can directly encode formulas with this syntax in text files to be used with the system [Genevès & Layaïda, 2006a]. This concrete syntax is used as a single unifying notation throughout all the chapter.

φ	::=		formula
		`T`	true
	|	`F`	false
	|	l	element name
	|	p	atomic proposition
	|	`#`	start context
	|	φ `|` φ	disjunction
	|	φ `&` φ	conjunction
	|	φ `=>` φ	implication
	|	φ `<=>` φ	equivalence
	|	`(`φ`)`	parenthesized formula
	|	$\tilde{\varphi}$	negation
	|	`<`p`>`φ	existential modality
	|	`<`l`>T`	attribute named l
	|	`$X`	variable
	|	`let` $\langle$`$X` $= \varphi\rangle^{\oplus}$ `in` φ	binder for recursion
	|	predicate	predicate (See Section 5.4)
p	::=		program inside modalities
		`1`	first child
	|	`2`	next sibling
	|	`-1`	parent
	|	`-2`	previous sibling

Figure 5.6: Concrete Syntax of Formulas.

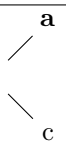
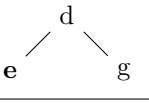
Sample Formula	Tree	XML
`a & <1>b`		`<a><b/></a>`
`a & <1>(b & <2>c)`		`<a><b/><c/></a>`
`e & <-1>(d & <2>g)`		`<d><e/></d><g/>`
`f & <-2>(g & ~<2>T)`	none	none

Figure 5.7: Sample Formulas and Satisfying Trees.

The semantics of logical formulas corresponds to the classical semantics of a μ-calculus interpreted over finite tree structures. A formula is satisfiable iff there exists a finite binary tree with attributes for which the formula holds at some node. This is formally defined in [Genevès et al., 2007b], and we review it informally below through a series of examples.

There is a difference between an element name and an atomic proposition[2]: an element has one and only one element name, whereas it can satisfy multiple atomic propositions. We use atomic propositions to attach specific information to tree nodes, not related to their XML labeling. For example, the start context (a reserved atomic proposition) is used to mark the starting context nodes for evaluating XPath expressions.

The logic uses modalities for navigating in binary trees. A modality `<`p`>`φ can be read as follows: "there exists a successor node by program p such that φ holds at this successor". As shown on Figure 5.6, a program p is simply one of the four basic programs $\{1, 2, -1, -2\}$. Program `1` allows navigating from a node down to its first successor, and program `2` allows navigating from a node down to its second successor. The logic also features converse programs `-1` and `-2` for navigating upward in binary trees, respectively from the first successor to its parent and from the second successor to its previous sibling. Table on Figure 5.7 gives some simple formulas using modalities for navigating in binary trees, together with sample satisfying trees, in binary and unranked tree representations.

The logic allows expressing recursion in trees through the recursive binder. For example the recursive formula:

```
let $X = b | <2>$X in $X
```

means that either the current node is named `b` or there is a sibling of the current node which is named `b`. For this purpose, the variable `$X` is bound to the

[2]In practice, an atomic proposition must start with a "_".

subformula `b | <2>$X` which contains an occurence of `$X` (therefore defining the recursion). The scope of this binding is the subformula that follows the "`in`" symbol of the formula, that is `$X`. The entire formula can thus be seen as a compact recursive notation for a infinitely nested formula of the form:

```
b | <2>(b | <2>(b | <2>(...)))
```

Recursion allows expressing global properties. For instance, the recursive formula:

```
~ let $X = a | <1>$X | <2>$X in $X
```

expresses the absence of nodes named `a` in the whole subtree of the current node (including the current node). Furthermore, the fixpoint operator makes possible to bind several variables at a time, which is specifically useful for expressing mutual recursion. For example, the mutually recursive formula:

```
let
  $X = (a & <2>$Y) | <1>$X | <2>$X,
  $Y = b | <2>$Y
in $X
```

asserts that there is a node somewhere in the subtree such that this node is named `a` and it has at least one sibling which is named `b`. Binding several variables at a time provides a very expressive yet succinct notation for expressing mutually recursive structural patterns (that are common in XML Schemas, for instance).

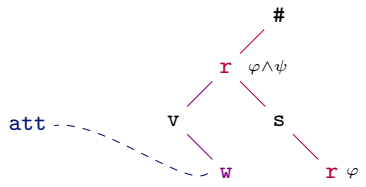
From a theoretical perspective, the recursive binder `let $X` $= \varphi$ `in` φ corresponds to the fixpoint operators of the μ-calculus. It is shown in [Genevès et al., 2007b] that the least fixpoint and the greatest fixpoint operators of the μ-calculus coincide over finite tree structures, for a restricted class of formulas called cycle-free formulas.

5.3.2 Queries

The logic is expressive enough to capture the set of XPath expressions presented in Section 5.2.3. For example, Figure 5.8 illustrates how the sample XPath expression:

```
child::r[child::w/@att]
```

is expressed in the logic. From a given context in an XML document, this expression selects all `r` child nodes which have at least one `w` child with an attribute `att`. Figure 5.8 shows how it is expressed in the logic, on the binary tree representation. The formula holds for `r` nodes which are selected by the expression. The first part of the formula, φ, corresponds to the step `child::r` which selects candidates `r` nodes. The second part, ψ, navigates downward in the subtrees of these candidate nodes to verify that they have at least one immediate `w` child with an attribute `att`.



Translated Query: `child::r[child::w/@att]`

Translation:

$$\underbrace{\texttt{r \& (let \$X=<-1>\# | <-2>\$X)}}_{\varphi}\ \underbrace{\texttt{\& <1>let \$Y=w \& <att>T | <2>\$Y}}_{\psi}$$

Figure 5.8: XPath Translation Example.

This example illustrates the need for converse programs inside modalities. The translated XPath expression only uses forward axes (child and attribute), nevertheless both forward and backward modalities are required for its logical translation. Without converse programs we would have been unable to differentiate selected nodes from nodes whose existence is simply tested. More generally, properties must often be stated on both the ancestors and the descendants of the selected node. Equipping the logic with both forward and converse programs is therefore crucial. Logics without converse programs may only be used for solving XPath emptiness but cannot be used for solving other decision problems such as containment efficiently.

A systematic translation of XPath expressions into the logic is given in [Genevès et al., 2007b]. In this chapter, we extended it to deal with attributes. We implemented a compiler that takes any expression of the fragment of Figure 5.2 and computes its logical translation. With the help of this compiler, we extend the syntax of logical formulas with a logical predicate `select`("*query*", φ). This predicate compiles the XPath expression *query* given as parameter into the logic, starting from a context that satisfies φ. The XPath expression to be given as parameter must match the syntax of the XPath fragment shown on Figure 5.2 (or Figure 5.3). In a similar manner, we introduce the predicate `exists`("*query*", φ) which tests the existence of *query* from a context satisfying φ, in a qualifier-like manner (without moving to its result). Additionally, the predicate `select`("*query*") is introduced as a shortcut for `select`("*query*", `#`), where `#` simply marks the initial context node of the XPath expression[3]. The predicate `exists`("*query*") is a shortcut for `exists`("*query*", `T`). These syntactic extensions of the logic allow the user to easily embed XPath expressions and formulate decision problems out of them (like e.g. containment or any other boolean combination). In the next sections we explain how the framework allows combining queries with schema

[3]This mark is especially useful for comparing two or more XPath expressions from the same context.

information for formulating problems.

5.3.3 Tree Types

Tree type expressions are compiled into the logic in two steps: the first stage translates them into binary tree type expressions, and the second step actually compiles this intermediate representation into the logic. The translation procedure from tree type expressions to binary tree type expressions is well-known and detailed in [Genevès, 2006]. The syntax of output expressions follows:

$\mathcal{T}$	::=		binary tree type expression
		$\emptyset$	empty set
	|	`()`	empty tree
	|	$\mathcal{T} \mid \mathcal{T}$	disjunction
	|	$l\,(\underline{\mathrm{a}})\,[x, x]$	element definition
	|	let $\overline{x_i.\mathcal{T}_i}$ in $\mathcal{T}$	binder

Attribute expressions are not concerned by this transformation to binary form: they are simply attached, unchanged, to new (binary) element definitions. Finally, binary tree type expressions are compiled into the logic. This translation step was introduced and proven correct in [Genevès et al., 2007b]. Originally, the translation takes a tree type expression $\mathcal{T}$ and returns the corresponding logical formula. Here, we extend it slightly but crucially: the logical translation of an expression $\mathcal{T}$ is given by the function $\mathrm{tr}(\mathcal{T})^{\psi}_{\varphi}$ defined below, that takes additional arguments φ and ψ:

$$\mathrm{tr}(\mathcal{T})^{\psi}_{\varphi} \stackrel{\text{def}}{=} \mathtt{F} \quad \text{for} \quad \mathcal{T} = \emptyset, \mathtt{()}$$

$$\mathrm{tr}(\mathcal{T}_1 \mid \mathcal{T}_2)^{\psi}_{\varphi} \stackrel{\text{def}}{=} \mathrm{tr}(\mathcal{T}_1)^{\psi}_{\varphi} \ \mathtt{|}\ \mathrm{tr}(\mathcal{T}_2)^{\psi}_{\varphi}$$

$$\mathrm{tr}(l\,(\underline{\mathrm{a}})\,[x_1, x_2])^{\psi}_{\varphi} \stackrel{\text{def}}{=} \left(l \ \mathtt{\&}\ \varphi \ \mathtt{\&}\ \mathrm{tra}(\underline{\mathrm{a}}) \ \mathtt{\&}\ \underline{\mathrm{succ}}_1(x_1) \ \mathtt{\&}\ \underline{\mathrm{succ}}_2(x_2)\right) \ \mathtt{|}\ \psi$$

$$\mathrm{tr}(\text{let } \overline{x_{ii}.\mathcal{T}_{ii}} \text{ in } \mathcal{T})^{\psi}_{\varphi} \stackrel{\text{def}}{=} \mathtt{let}\ \overline{\$\mathtt{X}_i = \mathrm{tr}(\mathcal{T}_i)^{\psi}_{\varphi}}\ \mathtt{in}\ \mathrm{tr}(\mathcal{T})^{\psi}_{\varphi}$$

The addition of φ and ψ (respectively in a new conjunction and a new disjunction) is a key element for the definition of predicates in Section 5.4. More precisely, this allows marking type sub-expressions so that they can be distinguished in predicates, as explained in Section 5.3.4. In addition, φ and ψ are either true, false, or simple atomic propositions. Thus, it is worth noticing that their addition does not affect the linear complexity of tree type translation. The function $\underline{\mathrm{succ}}\,(\cdot)$ describes the type for each successor:

$$\underline{\mathrm{succ}}_p(x) \quad = \quad \begin{cases} \mathtt{\tilde{}<}p\mathtt{>T} & \underline{\text{if } x \text{ is bound to } \mathtt{()}} \\ \mathtt{\tilde{}<}p\mathtt{>T\ |\ <}p\mathtt{>\$X} & \underline{\text{if nullable}(x)} \\ \mathtt{<}p\mathtt{>\$X} & \underline{\text{if not nullable}(x)} \end{cases}$$

according to the predicate $\underline{\mathrm{nullable}}(x)$ which indicates whether the type $T \neq \mathtt{()}$ bound to x contains the empty tree.

The function tra(a) compiles attribute expressions associated with element definitions as follows:

$$\begin{aligned}
\mathrm{tra}(\texttt{()}) &\stackrel{\text{def}}{=} \mathrm{notothers}(\texttt{()}) \\
\mathrm{tra}(\mathrm{list} \mid \mathrm{a}) &\stackrel{\text{def}}{=} \mathrm{tra}(\mathrm{list})\ \texttt{\&}\ \mathrm{notothers}(\mathrm{list}) \\
\mathrm{tra}(\mathrm{list}, \mathrm{list}') &\stackrel{\text{def}}{=} \mathrm{tra}(\mathrm{list})\ \texttt{\&}\ \mathrm{tra}(\mathrm{list}') \\
\mathrm{tra}(l\texttt{?}) &\stackrel{\text{def}}{=} l\ \texttt{|}\ \tilde{}\, l \\
\mathrm{tra}(l) &\stackrel{\text{def}}{=} l \\
\mathrm{tra}(\neg l) &\stackrel{\text{def}}{=} \tilde{}\, l
\end{aligned}$$

In usual schemas (e.g. DTDs, XML Schemas) when no attribute is specified for a given element, it simply means no attribute is allowed for the defined element. This convention must be explicitly stated in the logic. This is the role of the function "notothers(list)" which returns the negated disjunction of all attributes not present in list. As a result, taking attributes into account comes at an extra-cost. The above translation appends a (potentially very large) formula in which all attributes occur, for each element definition. In practice, a placeholder atomic proposition is inserted until the full set of attributes involved in the problem formulation is known. When the whole formula has been parsed, placeholders are replaced by the conjunction of negated attributes they denote. This extra-cost can be observed in practice, and the system allows two modes of operations: with or without attributes[4]. Nevertheless the system is still capable of handling real world DTDs (such as the DTD of XHTML 1.0 Strict) with attributes. This is due to (1) the limited expressive power of languages such as DTD that do not allow for disjunction over attribute expressions (like "list | a"); and, more importantly, (2) the satisfiability-testing algorithm which is implemented using symbolic techniques [Genevès et al., 2008].

Tree type expressions form the common internal representation for a variety of XML schema definition languages. In practice, the logical translation of a tree type expression $\mathcal{T}$ are obtained directly from a variety of formalisms for defining schemas, including DTD, XML Schema, and Relax NG. For this purpose, the syntax of logical formulas is extended with a predicate $\texttt{type}(\texttt{"}\cdot\texttt{"}, \cdot)$. The logical translation of an existing schema is returned by $\texttt{type}(\texttt{"}\mathrm{f}\texttt{"}, l)$ where f is a file path to the schema file and l is the element name to be considered as the entry point (root) of the given schema. Any occurrence of this predicate will parse the given schema, extract its internal tree type representation $\mathcal{T}$, compile it into the logic and return the logical formula $\mathrm{tr}(\mathcal{T})^{\mathtt{F}}_{\mathtt{T}}$.

5.3.4 Type Tagging

A tag (or "color") is introduced in the compilation of schemas with the purpose of marking all node types of a specific schema. A tag is simply a fresh atomic

[4]The optional argument "-attributes" must be supplied for attributes to be considered.

proposition passed as a parameter to the translation of a tree type expression. For example: $\mathrm{tr}(\mathcal{T})^{\mathtt{F}}_{\mathrm{xhtml}}$ is the logical translation of $\mathcal{T}$ where each element definition is annotated with the atomic proposition "xhtml". With the help of tags, it becomes possible to refer to the element types in any context. For instance, one may formulate $\mathrm{tr}(\mathcal{T})^{\mathtt{F}}_{\mathrm{xhtml}}$ | $\mathrm{tr}(\mathcal{T}')^{\mathtt{F}}_{\mathrm{smil}}$ for denoting the union of all $\mathcal{T}$ and $\mathcal{T}'$ documents, while keeping a way to distinguish element types; even if some element names are shared by the two type expressions.

Tagging becomes even more useful for characterizing evolutions between successive versions of a single schema. In this setting, we need a way to distinguish nodes allowed by a newer schema version from nodes allowed by an older version. This distinction must not be based only on element names, but also on content models. Assume for instance that $\mathcal{T}'$ is a newer version of schema $\mathcal{T}$. If we are interested in the set of trees allowed by $\mathcal{T}'$ but not allowed by $\mathcal{T}$ then we may formulate:

$$\mathrm{tr}(\mathcal{T}')^{\mathtt{F}}_{\mathtt{T}} \ \&\tilde{}\ \mathrm{tr}(\mathcal{T})^{\mathtt{F}}_{\mathtt{T}}$$

If we now want to check more fine-grained properties, we may rather be interested in the following (tagged) formulation:

$$\mathrm{tr}(\mathcal{T}')^{\mathtt{F}}_{\mathrm{all}} \ \&\tilde{}\ \mathrm{tr}(\mathcal{T})^{\tilde{}\,\mathrm{old\_complement}}_{\mathtt{T}}$$

In this manner, we can distinguish elements that were added in $\mathcal{T}'$ and whose names did not occur in $\mathcal{T}$, from elements whose names already occurred in $\mathcal{T}$ but whose content model changed in $\mathcal{T}'$, for instance.

In practice, a type is tagged using the predicate $\mathtt{type}(\mathtt{"f"}, l, \varphi, \varphi')$ which parses the specified schema, converts it into its logical representation $\mathcal{T}$ and returns the formula $\mathrm{tr}(\mathcal{T})^{\varphi'}_{\varphi}$. This kind of type tagging is useful for studying the consequences of schema updates over queries, as presented in the next sections.

5.4 Analysis Predicates

This section introduces the basic analysis tasks offered to XML application designers for assessing the impact of schema evolutions. In particular, we propose a means for identifying the precise reasons for type mismatches or changes in query results under type constraints.

For this purpose, we build on our query and type expression compilers, and define additional predicates that facilitate the formulation of decision problems at a higher level of abstraction. Specifically, these predicates are introduced as logical macros with the goal of allowing system usage while focusing (only) on the XML-side properties, and keeping underlying logical issues transparent for the user. Ultimately, we regard the set of basic logical formulas (such as modalities and recursive binders) as an assembly language, into which predicates are translated.

We illustrate this principle with two simple predicates designed for checking backward-compatibility of schemas, and query satisfiability in the presence of a schema.

- The predicate `backward_incompatible`$(\mathcal{T}, \mathcal{T}')$ takes two type expressions as parameters, and assumes $\mathcal{T}'$ is an altered version of $\mathcal{T}$. This predicate is unsatisfiable iff all instances of $\mathcal{T}'$ are also valid against $\mathcal{T}$. Any occurrence of this predicate in the input formula will automatically be compiled as $\mathrm{tr}(\mathcal{T}')^{\mathtt{F}}_{\mathtt{T}}$ `&~` $\mathrm{tr}(\mathcal{T})^{\mathtt{F}}_{\mathtt{T}}$.

- The predicate `non_empty`$(\texttt{"}\underline{query}\texttt{"}, \mathcal{T})$ takes an XPath expression (with the syntax defined on Figure 5.2) and a type expression as parameters, and is unsatisfiable iff the query always returns an empty set of nodes when evaluated on an XML document valid against $\mathcal{T}$. This predicate compiles into `select`$(\texttt{"}\underline{query}\texttt{"}, \mathrm{tr}(\mathcal{T})^{\mathtt{F}}_{\mathtt{T}}$ `& #`$)$ where the top-level predicate `select`$(\texttt{"}\underline{query}\texttt{"}, \varphi)$ compiles the XPath expression $\underline{query}$ into the logic, starting from a context that satisfies φ, as explained in Section 5.3.2. This can be used to check whether the modification of the schema does not contradict any part of the query.

Notice that the predicate `non_empty`$(\texttt{"}\underline{query}\texttt{"}, \mathcal{T})$ can be used for checking whether a query that is valid[5] against a schema remains valid with an updated version of a schema. In other terms, this predicate allows determining whether a query that must always return a non-empty result (whatever the tree on which it is evaluated) keeps verifying the same property with a new version of a schema.

A second, more-elaborate, class of predicates allows formulating problems that combine both a query $\underline{query}$ and two type expressions $\mathcal{T}, \mathcal{T}'$ (where $\mathcal{T}'$ is assumed to be a evolved version of $\mathcal{T}$):

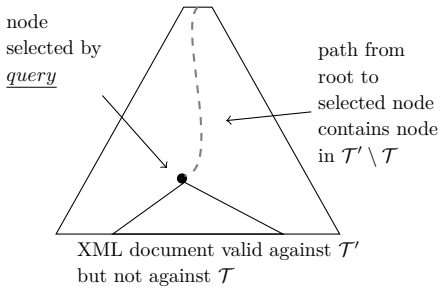
- `new_element_name`$(\texttt{"}\underline{query}\texttt{"}, \mathcal{T}, \mathcal{T}')$ is satisfied iff the query $\underline{query}$ selects elements whose names did not occur at all in $\mathcal{T}$. This is especially useful for queries whose last navigation step contains a "`*`" node test and may thus select unexpected elements. This predicate is compiled into:

$$\texttt{~element}(\mathcal{T})\ \texttt{\&}\ \texttt{select}(\texttt{"}\underline{query}\texttt{"}, \mathrm{tr}(\mathcal{T}')^{\mathtt{F}}_{\mathtt{T}})$$

  where `element`$(\mathcal{T})$ is another predicate that builds the disjunction of all element names occuring in $\mathcal{T}$. In a similar manner, the predicate `attribute`(φ) builds the logical disjunction of all attribute names used in φ.

- `new_region`$(\texttt{"}\underline{query}\texttt{"}, \mathcal{T}, \mathcal{T}')$ is satisfied iff the query $\underline{query}$ selects elements whose names already occurred in $\mathcal{T}$, but such that these nodes now occur in a new context in $\mathcal{T}'$. In this setting, the path from the root of the document to a node selected by the XPath expression $\underline{query}$ contains a node whose type is defined in $\mathcal{T}'$ but not in $\mathcal{T}$ as illustrated below:

[5] We say that a query is $\underline{\text{valid}}$ iff its negation is unsatisfiable.



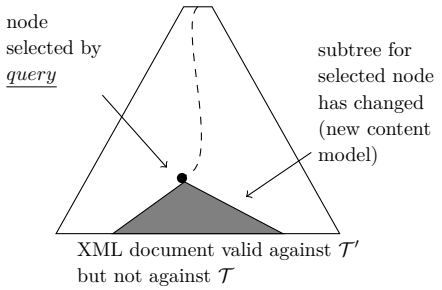
The predicate `new_region`("*query*", $\mathcal{T}, \mathcal{T}'$) is logically defined as follows:

$$
\begin{aligned}
&\texttt{new\_region}(\text{"}\underline{query}\text{"}, \mathcal{T}, \mathcal{T}') \stackrel{\text{def}}{=} \\
&\quad \texttt{select}(\text{"}\underline{query}\text{"}, \mathrm{tr}(\mathcal{T}')^{\texttt{F}}_{\texttt{\_all}} \ \texttt{\&}\tilde{}\ \mathrm{tr}(\mathcal{T})^{\tilde{}}_{\texttt{T}}{}^{\texttt{-old\_complement}}) \\
&\qquad \texttt{\&}\tilde{}\ \texttt{added\_element}(\mathcal{T}, \mathcal{T}') \\
&\qquad \texttt{\&}\ \texttt{ancestor}(\texttt{\_old\_complement}) \\
&\qquad \texttt{\&}\tilde{}\ \texttt{descendant}(\texttt{\_old\_complement}) \\
&\qquad \texttt{\&}\tilde{}\ \texttt{following}(\texttt{\_old\_complement}) \\
&\qquad \texttt{\&}\tilde{}\ \texttt{preceding}(\texttt{\_old\_complement})
\end{aligned}
$$

The previous definition heavily relies on the partition of tree nodes defined by XPath axes, as illustrated by Figure 5.9.

The definition of `new_region`("*query*", $\mathcal{T}, \mathcal{T}'$) uses `added_element`($\mathcal{T}, \mathcal{T}'$), an auxiliary predicate, that builds the disjunction of all element names defined in $\mathcal{T}'$ but not in $\mathcal{T}$ (or in other terms, elements that were added in $\mathcal{T}'$). In a similar manner, the predicate `added_attribute`(φ, φ') builds the disjunction of all attribute names defined in $\mathcal{T}'$ but not in $\mathcal{T}$. The predicate `new_region`("*query*", $\mathcal{T}, \mathcal{T}'$) is useful for checking whether a query selects a different set of nodes with $\mathcal{T}'$ than with $\mathcal{T}$ because selected elements may occur in new regions of the document due to changes brought by $\mathcal{T}'$.

- `new_content`("*query*", $\mathcal{T}, \mathcal{T}'$) is satisfied iff the query *query* selects elements whose names were already defined in $\mathcal{T}$, but whose content model has changed due to evolutions brought by $\mathcal{T}'$, as illustrated below:



The definition of `new_content`("*query*", $\mathcal{T}, \mathcal{T}'$) follows:

$$
\begin{aligned}
&\texttt{new\_content}(\texttt{"}\underline{query}\texttt{"}, \mathcal{T}, \mathcal{T}') \stackrel{\text{def}}{=} \\
&\qquad \texttt{select}(\texttt{"}\underline{query}\texttt{"}, \mathrm{tr}(\mathcal{T}')^{\texttt{F}}_{\texttt{\_all}} \mathbin{\texttt{\&\~{}}} \mathrm{tr}(\mathcal{T})^{\sim}_{\texttt{T}}{}^{\texttt{\_old\_complement}}) \\
&\qquad\qquad \texttt{\&\~{}added\_element}(\mathcal{T}, \mathcal{T}') \\
&\qquad\qquad \texttt{\&\~{}ancestor}(\texttt{added\_element}(\mathcal{T}, \mathcal{T}')) \\
&\qquad\qquad \texttt{\& descendant}(\texttt{\_old\_complement}) \\
&\qquad\qquad \texttt{\&\~{}following}(\texttt{\_old\_complement}) \\
&\qquad\qquad \texttt{\&\~{}preceding}(\texttt{\_old\_complement})
\end{aligned}
$$

The predicate `new_content`("*query*", $\mathcal{T}, \mathcal{T}'$) can be used for ensuring that XPath expressions will not return nodes with a possibly new content model that may cause problems. For instance, this allows checking whether an XPath expression whose resulting node set is converted to a string value (as in, e.g. XPath expressions used in XSLT "value-of" instructions) is affected by the changes from $\mathcal{T}$ to $\mathcal{T}'$.

- `new_sibling`("*query*", $\mathcal{T}, \mathcal{T}'$) is satisfied iff the query *query* selects elements whose names already occurred in $\mathcal{T}$, but such that they now occur with new potential siblings due to $\mathcal{T}'$. The notion of context, here, is extended to be not only the chain of ancestors from the selected node to the root but also the set of previous and following siblings of the selected node.

Figure 5.9: XPath axes: partition of tree nodes.

The previously defined predicates can be used to help the programmer identify precisely how type constraint evolutions affect queries. They can even be combined with usual logical connectives to formulate even more sophisticated problems. For example, let us define the predicate `exclude`(φ) which is satisfiable iff there is no node that satisfies φ in the whole tree. This predicate can be used for excluding specific element names or even nodes selected by a given XPath expression. It is defined as follows:

$$\texttt{exclude}(\varphi) \stackrel{\text{def}}{=} \tilde{}\,\texttt{ancestor-or-self}(\texttt{descendant-or-self}(\varphi))$$

This predicate can also be used for checking properties in an iterative manner, refining the property to be tested at each step. It can also be used for verifying fine-grained properties. For instance, one may check whether $\mathcal{T}'$ defines the same set of trees as $\mathcal{T}$ modulo new element names that were added in $\mathcal{T}'$ with the following formulation:

$$\tilde{}\,(\mathcal{T}\ \texttt{<=>}\ \mathcal{T}')\ \texttt{\&}\ \texttt{exclude}(\texttt{added\_element}(\mathcal{T}, \mathcal{T}'))$$

This allows identifying that, during the type evolution from $\mathcal{T}$ to $\mathcal{T}'$, the query results change has not been caused by the type extension but by new compositions of nodes from the older type.

In practice, instead of taking internal tree type representations (as defined in Section 5.2.2) as parameters, most predicates do actually take any logical formula as parameter, or even schema paths as parameters. We believe this facilitates predicates usage and, most notably, how they can be composed together. Figure 5.10 gives the syntax of built-in predicates as they are implemented in the system, where f is a file path to a DTD (.dtd), XML Schema (.xsd), or Relax NG (.rng). In addition of aforementioned predicates, the predicate `descendant`(φ) forces the existence of a node satisfying φ in the subtree, and predicate-name($\langle\varphi\rangle^{\oplus}$) is a call to a custom predicate, as explained in the next section.

5.4.1 Custom Predicates

Following the spirit of predicates presented in the previous section, users may also define their own custom predicates. The full syntax of XML logical specifications to be used with the system is defined on Figure 5.11, where the meta-syntax $\langle X\rangle^{\oplus}$ means one or more occurrence of X separated by commas. A global problem specification can be any formula (as defined on Figure 5.6), or a list of custom predicate definitions separated by semicolons and followed by a formula. A custom predicate may have parameters that are instanciated with actual formulas when the custom predicate is called (as shown on Figure 5.10). A formula bound to a custom predicate may include calls to other predicates, but not to the currently defined predicate (recursive definitions must be made through the let binder shown on Figure 5.6).

$$
\begin{array}{lll}
\underline{\text{predicate}} & ::= & \\
 & & \texttt{select}(\text{"}\underline{query}\text{"}) \\
 & | & \texttt{select}(\text{"}\underline{query}\text{"}, \varphi) \\
 & | & \texttt{exists}(\text{"}\underline{query}\text{"}) \\
 & | & \texttt{exists}(\text{"}\underline{query}\text{"}, \varphi) \\
 & & \\
 & | & \texttt{type}(\text{"}\underline{\text{f}}\text{"}, l) \\
 & | & \texttt{type}(\text{"}\underline{\text{f}}\text{"}, l, \varphi, \varphi') \\
 & | & \texttt{forward\_incompatible}(\varphi, \varphi') \\
 & | & \texttt{backward\_incompatible}(\varphi, \varphi') \\
 & & \\
 & | & \texttt{element}(\varphi) \\
 & | & \texttt{attribute}(\varphi) \\
 & | & \texttt{descendant}(\varphi) \\
 & | & \texttt{exclude}(\varphi) \\
 & | & \texttt{added\_element}(\varphi, \varphi') \\
 & | & \texttt{added\_attribute}(\varphi, \varphi') \\
 & & \\
 & | & \texttt{non\_empty}(\text{"}\underline{query}\text{"}, \varphi) \\
 & | & \texttt{new\_element\_name}(\text{"}\underline{query}\text{"}, \text{"}\underline{\text{f}}\text{"}, \text{"}\underline{\text{f}'}\text{"}, l) \\
 & | & \texttt{new\_region}(\text{"}\underline{query}\text{"}, \text{"}\underline{\text{f}}\text{"}, \text{"}\underline{\text{f}'}\text{"}, l) \\
 & | & \texttt{new\_sibling}(\text{"}\underline{query}\text{"}, \text{"}\underline{\text{f}}\text{"}, \text{"}\underline{\text{f}'}\text{"}, l) \\
 & | & \texttt{new\_content}(\text{"}\underline{query}\text{"}, \text{"}\underline{\text{f}}\text{"}, \text{"}\underline{\text{f}'}\text{"}, l) \\
 & | & \underline{\text{predicate-name}}(\langle \varphi \rangle^{\oplus})
\end{array}
$$

Figure 5.10: Syntax of Predicates for XML Reasoning.

5.5 Impact of Standard Schema Evolution on Valid Documents

As depicted on Fig. 5.1, the whole system relies on a satisfiability solver for the underlying logic. The main principe of the satisfiability-solver is an exhaustive search for a tree that satisfies the formula. The search relies on a least fixpoint computation that starts from all possible leaves and attempt to plug every possible parent node at each further step. The algorithm terminates once the initial formula has been found to hold in a given node of the tree. Otherwise, the

$$
\begin{array}{lllll}
\underline{\text{spec}} & ::= & & & \\
 & & \varphi & & \text{formula (see Fig. 5.6)} \\
 & | & \underline{\text{def}}; \varphi & & \\
\underline{\text{def}} & ::= & & & \\
 & & \underline{\text{predicate-name}}(\langle l \rangle^{\oplus}) = \varphi' & & \text{custom definition} \\
 & | & \underline{\text{def}}; \underline{\text{def}} & & \text{list of definitions}
\end{array}
$$

Figure 5.11: Global Syntax for Specifying Problems.

algorithm terminates when no more parent nodes can be added. The algorithm, as well as proofs of its soundness and completeness, optimal complexity, and implementation techniques are detailed in [Genevès et al., 2007b].

We have carried out extensive experiments of the system in real world settings, e.g. with popular web schemas such as XHTML, MathML, SVG, SMIL (Table on Figure 5.12 gives details related to their respective sizes). In this section, we show how the tool can be used to analyze different situations where schemas changes have important consequences on the validity of existing documents.

Schema	Variables	Elements	Attributes
XHTML 1.0 basic DTD	71	52	57
XHTML 1.1 basic DTD	89	67	83
MathML 1.01 DTD	137	127	72
MathML 2.0 DTD	194	181	97

Figure 5.12: Sizes of (Some) Considered Schemas.

One major role of organizations such as W3C is to contribute to the standardization effort leading to a unique widely accepted set of constraints for a given class of documents. Designing a normative specification is a complex process, which is made even harder by a few important considerations. For example, when a language is designed, one need to take into account how future versions of that language can evolve. For a particular version of a language, not only the schema constraints allowed by that version need to be considered but also how they can be modified in future versions. This allows to address how an implementation of this version should process document variants added by future schema versions.

Specifically, we identify three different properties for a specification:

- Forward compatibility: All instances of an older specification should be valid with respect to newer specifications. This ensures that a document can still be processed properly with applications implementing newer specifications.
- Backward compatibility without added elements/attributes: New combinations of old elements are not supposed to be introduced in later specifications. Otherwise, an application implementing an older specification will not able to process a document that conforms to some future specification, even if this document does not contain any element or attribute introduced as extensions.
- Equivalence between schema versions: A given specification can be expressed in a variety of schema definition languages like DTD, XML Schema, Relax NG. We expect the different schema versions of the same specification to define the same set of documents modulo the expressivity of the schema language [Murata et al., 2005].

An XML schema definition (whether normative or not) often evolves over time, as new needs often result in new features usually introduced as new elements and attributes. However we believe that this normal evolution should not break the three previous properties.

We report below on using the framework for characterizing the evolution of the main standard document formats used on the web, including W3C XHTML, SMIL, SVG and MathML, based on the criteria identified above. This kind of analyses yield important observations on the validity of, potentially, billions of documents.

5.5.1 XHTML Basic

The first test consists in analyzing the relationship (forward and backward compatibility) between XHTML basic 1.0 and XHTML basic 1.1 schemas. In particular, backward compatibility can be checked by the following command:

```
backward_incompatible("xhtml-basic10.dtd",
                      "xhtml-basic11.dtd", "html")
```

Executing the test yields a counter example as the new schema contains new element names. The counter example (shown below) contains a `style` element occurring as a child of `head`, which is not permitted in XHTML basic 1.0:

```
<html>
  <head>
    <title/>
    <style type="_otherV"/>
  </head>
  <body/>
</html>
```

The next step consists in focusing on the relationship between both schemas excluding these new elements. This can be formulated by the following command:

```
backward_incompatible("xhtml-basic10.dtd",
                      "xhtml-basic11.dtd", "html")
& exclude(added_element(
      type("xhtml-basic10.dtd","html"),
      type("xhtml-basic11.dtd", "html")))
```

The result of the test shows a counter example document that proves that XHTML basic 1.1 is not backward compatible with XHTML basic 1.0 even if new elements are not considered. In particular, the content model of the `label` element cannot have an `a` element in XHTML basic 1.0 while it can in XHTML basic 1.1. The counter example produced by the solver is shown below:

```
<html>
  <head>
    <object>
      <label>
        <a href="...">
          <img/>
        </a>
        <img/>
      </label>
      <param/>
    </object>
    <meta/>
    <title/>
    <base/>
  </head>
  <body/>
</html>

XTML basic 1.0 validity error: element a is not
declared in label list of possible children
```

5.5.2 SMIL

The second test consists in analyzing the relationship (forward and backward compatibility) between several versions of the SMIL standard[6], namely versions 1.0, 2.0, and 3.0. In particular, forward compatibility between 1.0 and 2.0 can be checked by the following command:

```
forward_incompatible("SMIL10.dtd", "SMIL20.dtd", "smil")
```

The result of the test shows a counter example document that proves that there exist valid SMIL 1.0 documents that are not valid anymore with respect to SMIL 2.0. In fact that is because the content model of the `layout` element is defined as `any` in SMIL 1.0, whereas it is more restricted in SMIL 2.0. We observe that introducing `any` is a choice that has important consequences. Indeed, a document that was playable with 1.0 implementations may no longer be playable using 2.0 implementations. The counter example produced by the solver is shown below:

```
<smil>
  <head>
    <layout>
      <meta content="_otherV" name="_otherV"/>
    </layout>
  </head>
</smil>
```

[6]The first author was a member of the W3C SMIL working group and a co-author of SMIL 2.0 and 2.1.

```
SMIL 2.0 validity error:
Element layout content does not follow the DTD,
expecting (region|topLayout|root-layout|regPoint)*,
got (meta)
```

The lesson here is that introducing very permissive content models (like `any`) has to be considered very seriously. Indeed, that means that all future versions of the standard should be at least as permissive. Otherwise, all content produced with earlier (more permissive) versions becomes at risk. Therefore, the initial content model has to be carefully designed in order to avoid such situations.

The following example is even worse. We check forward compatibility between SMIL 2.0 and 3.0:

```
forward_incompatible("SMIL20.dtd",
                          "SMIL30Language.dtd", "smil")
```

We obtain the following counter-example:

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  <body>
    <switch>
      <animateMotion/>
    </switch>
    <a href="..."/>
  </body>
</smil>
```

This document is valid with respect to SMIL 2.0. However it does not validate with respect to SMIL 3.0. That is because the content model for the `switch` element was set to a more restrictive pattern in version 3.0 compared to 2.0, as the following validation error message suggests:

```
SMIL 3.0 validity error :
Element switch content does not follow the DTD,
expecting ((metadata | switch)* , ((((animate | set |
animateMotion | animateColor) , (metadata | switch)*)* ,
(((par | seq | excl | audio | video | animation | text |
 ... switch)*)+)) | (layout , (metadata | switch)*)*)),
got (animateMotion)
```

Now we would like to know if the bug is limited to the occurrence of the `animateMotion` element or whether it is more general. To this end, we progressively exclude elements named `animateMotion`, `set`, `animateColor`, and `animate`, as follows:

```
forward_incompatible("SMIL20.dtd",
                          "SMIL30Language.dtd", "smil")
  & exclude(animateMotion) & exclude(set)
  & exclude(animateColor) & exclude(animate)
```

We still obtain the following counter-example (valid w.r.t SMIL 2.0 but not w.r.t SMIL 3.0), which shows that the forward incompatibility is not limited to the occurence of the previous elements, but rather, to severe limitations of the `switch` content model introduced in 3.0. In other words, `switch` is an element which undermines SMIL forward compatibility.

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  <body>
    <switch>
      <seq/>
      <area/>
    </switch>
    <switch/>
    <a href="..."/>
  </body>
</smil>
```

5.5.3 SVG

The SVG test consists in analyzing the relationship (forward and backward compatibility) between SVG 1.0 et 1.1. In particular, we examine the different profiles (tiny, basic and full) from 1.0 and compare them to 1.1 schemas. Backward compatibility can be checked by the following command:

```
forward_incompatible("svg10.dtd",
              "svg11-flat-20030114.dtd", "svg")
```

The test is unsatisfiable meaning that SVG 1.1 is formally proven to be forward compatible with SVG 1.0. This is good news as it means that all 1.0 documents will be supported with 1.1 conforming implementations, without any exception. In the case where a 1.0 document does not play with a 1.1 implementation, this indicates a bug in the implementation and not in the SVG specification.

We observe here that the common practice of including a single doctype declaration within a document is questionable, since a document is not only valid w.r.t a given schema but also w.r.t to all future forward-compatible versions. Keeping track of this mapping between a document and several schemas allows the document to be supported by a larger set of implementations.

Similar tests on the SVG 1.1 tiny, basic and full also exhibit good results. This corresponds to the definition of these three profiles as strict subsets of each other. Furthermore, we believe that the use of a modularized version of a schema (as opposed to a complete redefinition) has helped in avoiding compatibility problems.

We now focus on testing the backward compatibility between the SVG basic 1.1 profile and SVG 1.0 profile. The test fails even if new features are left aside:

```
backward_incompatible("svg10.dtd",
            "svg11-basic.dtd", "svg")
& exclude( added_element(type("svg10.dtd", "svg"),
                         type("svg11-basic.dtd","svg")))
& exclude(switch)
```

This test yields the following counter-example which confirms that there is actually a flaw in the 1.1 specification:

```
<svg>
  <image href="..." width="..." height="...">
    <title/>
    <title/>
  </image>
</svg>
```

as it allows two `title` elements to occur inside an `image` element, which was not allowed in the 1.0.

5.5.4 MathML

We apply a similar investigation approach to MathML 1.0 and its newer version 2.0. We formulate a backward compatibility test without elements that were added in version 2.0. Furthermore, we want to exclude immediate trivial counter-examples involving the use of the `declare` element as well as of the `math` element occuring within the `annotation-xml` element. For this purpose, we use the following formulation:

```
backward_incompatible("mathml.dtd","mathml2.dtd","math")
& exclude( added_element( type("mathml.dtd","math"),
                          type("mathml2.dtd","math")))
& exclude(declare)
& (~descendant(math))
```

that bans the `declare` element from occuring in the whole tree (achieved with the use of the `exclude(declare)` predicate), and prevents the `math` element from ocurring in the root's subtree (owing to the use of the `(~descendant(math))` predicate) The following counter-example is produced:

```
<math>
  <apply>
    <annotation-xml>
      <mprescripts/>
    </annotation-xml>
  </apply>
</math>
```

Such backward incompatibilities suggest that applications cannot simply ignore new elements from newer schemas, as the combination of older elements may evolve significantly from one version to another.

5.6 Impact of Schema Evolution on Queries

In this section, we report on using the framework in order to evaluate the consequences of schema changes on XPath queries such as the ones found in transformations like the MathML content to presentation conversion [Pietriga, 2005].

5.6.1 MathML Content to Presentation Conversion

MathML is an XML format for describing mathematical notations and capturing both its mathematical structure and graphical rendering, also known as Content MathML and Presentation MathML respectively. The structure of a given equation is kept separate from the presentation and the rendering part can be generated from the structure description. This operation is usually carried out using an XSLT transformation that achieves the conversion. In this test series, we focus on the analysis of the queries contained in such a transformation sheet and evaluate the impact of the schema change from MathML 1.0 to MathML 2.0 on these queries.

Most of the queries contained in the transformation represent only a few patterns very similar up to element names. The following three patterns are the most frequently used:

```
Q1:  //apply[*[1][self::eq]]
Q2:  //apply[*[1][self::apply]/inverse]
Q3:  //sin[preceding-sibling::*[position()=last()
              and (self::compose or self::inverse)]]
```

The first test is formulated by the following command:

```
new_region("Q1","mathml.dtd","mathml2.dtd","math")
```

The result of the test shows a counter example document that proves that the query may select nodes in new contexts in MathML 2.0 compared to MathML 1.0. In particular, the query `Q1` selects `apply` elements whose ancestors can be `declare` elements, as indicated on the document produced by the solver[7]:

```
<math xmlns:solver="http://wam.inrialpes.fr/xml"
            solver:context="true">
  <declare>
    <apply solver:target="true">
      <eq/>
```

[7] Notice that the solver automatically annotates a pair of nodes related by the query: when the query is evaluated from a node marked with the attribute `solver:context`, the node marked with `solver:target` is selected.

```
    </apply>
    <condition/>
  </declare>
</math>
```

To evaluate the effect of this change, the counter example is filled with content and passed as an input parameter to the transformation. This shows immediately a bug in the transformation as the resulting document is not a MathML 2.0 presentation document. Based on this analysis, we know that the XSLT template associated with the match pattern `Q1` must be updated to cope with MathML evolution from version 1.0 to version 2.0.

The next test consists in evaluating the impact of the MathML type evolution for the query `Q2` while excluding all new elements added in MathML 2.0 from the test. This identifies whether old elements of MathML 1.0 can be composed in MathML 2.0 in a different manner. This can be performed with the following command:

```
new_content("Q2","mathml.dtd","mathml2.dtd","math")
& exclude(added_element(type("mathml.dtd","math"),
                        type("mathml2.dtd", "math")))
```

The test result shows an example document that effectively combines MathML 1.0 elements in a way that was not allowed in MathML 1.0 but permitted in MathML 2.0.

```
<math xmlns:solver="http://wam.inrialpes.fr/xml"
            solver:context="true">
  <apply solver:target="true">
    <apply>
      <inverse/>
    </apply>
    <annotation-xml>
      <math/>
    </annotation-xml>
    <condition/>
  </apply>
</math>
```

Similarly, the last test consists in evaluating the impact of the MathML type evolution for the query `Q3`, excluding all new elements added in MathML 2.0 and counter example documents containing `declare` elements (to avoid trivial counter examples):

```
new_region("Q3","mathml.dtd","mathml2.dtd","math")
& exclude(added_element(type("mathml.dtd","math"),
                        type("mathml2.dtd","math")))
                        & exclude(declare)
```

The counter example document shown below illustrates a case where the `sin` element occurs in a new context.

```
<math xmlns:solver="http://wam.inrialpes.fr/xml"
            solver:context="true">
  <apply>
    <annotation-xml>
      <math>
        <apply>
          <inverse/>
          <sin solver:target="true"/>
        </apply>
      </math>
    </annotation-xml>
  </apply>
</math>
```

Applying the transformation on previous examples yields documents which are neither MathML 1.0 nor MathML 2.0 valid. As a result, the stylesheet cannot be used safely over documents of the new type without modifications. In addition, the required changes to the stylesheet are not limited to the addition of new templates for MathML 2.0 elements. The templates that deal with the composition of MathML 1.0 elements should be revised as well.

5.7 System Implementation

We have implemented the whole software architecture described in Section 5.2 and illustrated on Figure 5.1. The tool presented in the previous chapter in Section 4.5.5 was extended to support the predicates language presented here.

All the previous tests were processed in less than 30 seconds on an ordinary laptop computer running Mac OS X. The 30s correspond to the most complex use cases. Most complex means analyzing recursive forward/backward and qualified queries such as `Q3`, under evolution of large and heavily recursive schemas such as XHTML and MathML (large number of type variables, elements and attributes: see Table on Figure 5.12). These are the hardest cases measured in practice with the implementation. Most of other schemas and queries usually found in applications are much simpler than the ones presented in this chapter and will obviously be solved much faster. Given the variety of schemas occurring in practice, we focused on the most complex W3C standard schemas. The full online implementation [Genevès & Layaïda, 2006a] allows to run all the tests described in the chapter as well as user-supplied ones. It shows intermediate compilation stages, generated formulae (in particular the translation of schemas into the logic), and reports on the performance of each step of the analysis.

5.8 Related Work

Schema evolution is an important topic and has been extensively explored in the context of relational, object-oriented, and XML databases. Most of the previous work for XML query reformulation is approached through reductions to relational problems [Beyer et al., 2005]. This is because schema evolution was considered as a storage problem where the priority consists in ensuring data consistency across multiple relational schema versions. In such settings, two distinct schemas and an explicit description of the mapping between them are assumed as input. The problem then consists in reformulating a query expressed in terms of one schema into a semantically equivalent query in terms of the other schema: see [Yu & Popa, 2005] and more recently [Moon et al., 2008] with references thereof.

In addition to the fundamental differences between XML and the relational data model, in the more general case of XML processing, schemas constantly evolve in a distributed, independent, and unpredictable environment. The relations between different schemas are not only unknown but hard to track. In this context, one priority is to help maintaining query consistency during these evolutions, which is still considered as a challenging problem [Sedlar, 2005; Rose, 2004]. The absence of evolution analysis tools for XML/XPath contrasts with the abundance of tools and methods routinely used in relational databases.

The work found in [Moro et al., 2007] discusses the impact of evolving XML schemas on query reformulation. Based on a taxonomy of XML schema changes during their evolution, the authors provide informal – not exact nor systematic – guidelines for writing queries which are less sensitive to schema evolution. In fact, studying query reformulation requires at least the ability to analyze the relationship between queries. For this reason, a closely related work is the problem of determining query containment and satisfiability under type constraints [Benedikt et al., 2005; Colazzo et al., 2006; Genevès et al., 2007b]. These static analysis tasks are also notably useful for performing query optimization [Groppe et al., 2006].

The works found in [Benedikt et al., 2005; Groppe & Groppe, 2008] study the complexity of XPath emptiness and containment for various fragments with or without type constraints (see [Benedikt & Koch, 2009] and references thereof for a survey). In [Colazzo et al., 2004, 2006], a technique is presented for statically ensuring correctness of paths. The approach deals with emptiness of XPath expressions without reverse axes. The work presented in [Genevès et al., 2007b] solves the more general problem of containment, including reverse axes.

The main distinctive idea pursued in this chapter is to develop a logical approach for guiding schema and query evolution. In contrast to the previous use of logics for proving properties such as query emptiness or equivalence, the goal here is different in that we seek to provide the necessary tools to produce relevant knowledge when such relations do not hold. From a complexity point-

of-view, it is worth noticing that the addition of predicates does not increase complexity for the underlying logic shown in [Genevès et al., 2007b].

We would also like to emphasize that, to the best of our knowledge, this work is the first to provide precise analyses of XML evolution, that was tested on real life use cases (such as XHTML and MathML types) and complex queries (involving recursive and backward navigation). As a consequence, in this context, analysis tools such as type-checkers [Hosoya & Pierce, 2003; Benzaken et al., 2003; Møller & Schwartzbach, 2005; Gapeyev et al., 2006; Castagna & Nguyen, 2008] do no match the expressiveness, typing precision, and analysis capabilities of the work presented here.

5.9 Conclusion

In this article, we present an application of a unifying logical framework for verifying forward/backward compatibility issues caused by schemas evolution. We provide evidence that such a framework can be successfully used to overcome the obstacles of the analysis of XML schema evolution. This kind of analyses is widely considered as a challenging problem in XML programming. As mentioned earlier, the difficulty is twofold: first it requires dealing with large and complex language constructions such as XML types and XPath queries, and second, it requires modeling and reasoning about evolution of such constructions.

We presented the logical foundations of the framework. We then applied the framework for analyzing two major issues due to schema evolution: first, the consequence on the validity of documents and, second, the impact on queries. The presented system detected several compatibility problems in the main document formats used on the web. The same tool also allows XML designers to identify queries that need reformulation in order to produce the expected results across successive schema versions. With this tool designers can examine precisely the impact of schema changes over queries, therefore facilitating their reformulation.

We gave illustrations of how to use the tool for schema evolution on realistic examples. In particular, we considered typical situations in applications involving evolution of W3C schemas used on the web such as XHTML and MathML. We believe that the tool can be very useful for standard schema writers and maintainers in order to assist them enforce some level of quality assurance on compatibility between versions.

One direction for future work is to search for techniques giving suggestions on how to rewrite the query into an equivalent one to accommodate schema changes.

Six

Functions, Polymorphism and Subtyping

Abstract

We consider a type algebra equipped with recursive, product, function, intersection, union, and complement types together with type variables and implicit universal quantification over them. We consider the subtyping relation recently defined by Castagna and Xu over such type expressions and show how this relation can be decided in EXPTIME, answering an open question. The novelty, originality and strength of our solution reside in introducing a logical modeling for the semantic subtyping framework. We model semantic subtyping in a tree logic and use a satisfiability-testing algorithm in order to decide subtyping. We report on practical experiments made with a full implementation of the system. This provides a powerful polymorphic type system aiming at maintaining full static type-safety of functional programs that manipulate trees, even with higher-order functions, which is particularly useful in the context of XML.

6.1 Introduction

This chapter studies parametric polymorphism for type systems aiming at maintaining full static type-safety of functional programs manipulating linked structures such as trees, potentially with higher-order functions. We consider a type algebra equipped with recursive, product, function (arrow), intersection, union, and complement types. We first show how the subtyping relation between such type expressions can be decided through a logical approach. Our main result solves an open problem: we prove the decidability of the subtyping relation when this type algebra is extended with type variables. This provides a powerful polymorphic type system (using ML-style prenex polymorphism, where variables are implicitly universally quantified at toplevel), for which defining the subtyping relation is not obvious, as pointed out in [Castagna & Xu, 2011] and discussed in Section 6.5.1, and for which no candidate definition of subtyping had been proved decidable before. The novelty, originality and strength of our solution reside in introducing a logical modeling for the semantic subtyping framework. Specifically, we model semantic subtyping in a mu-calculus over finite trees and rely on a satisfiability solver in order to decide subtyping in practice. We obtain an EXPTIME complexity bound as well as an efficient implementation in practice.

6.1.1 The Need for Polymorphism and Subtyping

Subtyping makes it possible to prove that term substitution in a program source code preserves type-safety. For example, let us consider a simple property relating polymorphic types of functions that manipulate lists. We consider a type α, and denote by $[\alpha]$ the type of α-lists (lists whose elements are of type α). The type τ of functions that process an α-list and return a boolean is written as follows:

$$\tau = [\alpha] \to \texttt{Bool} \tag{6.1}$$

where $\texttt{Bool} = \{\texttt{true}, \texttt{false}\}$ is the type containing only the two values `true` and `false` Now let us consider functions that distinguish α-lists of even length from α-lists of odd length: such a function returns `true` for lists with an even number of elements of type α, and returns `false` for lists with an odd number of elements of type α. One may represent the set of these functions by a type τ' written as follows:

$$\begin{array}{lll} & \text{even}[\alpha] & \to \{\texttt{true}\} \\ \wedge & \text{odd}[\alpha] & \to \{\texttt{false}\} \end{array} \tag{6.2}$$

where `{true}` and `{false}` are singleton types (containing just one value). If we make explicit the parametric types even$[\alpha]$ and odd$[\alpha]$, τ' becomes:

$$\begin{array}{lll} \tau' = & \mu v.(\alpha \times (\alpha \times v)) \vee \text{nil} & \to \{\texttt{true}\} \\ \wedge & \mu v.(\alpha \times (\alpha \times v)) \vee (\alpha \times \text{nil}) & \to \{\texttt{false}\} \end{array} \tag{6.3}$$

where $\times$ denotes the cartesian product, μ binds the variable v for denoting a recursive type, and nil is a singleton type.

Obviously, a particular function of type τ' can also be seen as a less-specific function of type τ. In other terms, from a practical point of view, a function of type τ can be replaced by a more specific function of type τ' while preserving type-safety (however the converse is not true). This is further formalized by the notion of subtyping; in that case we write:

$$\tau' \leqslant \tau \tag{6.4}$$

where $\leqslant$ denotes a subtyping relation that can be defined in two fundamentally different ways in the literature: either syntactically or semantically. In this chapter, we define $\leqslant$ as a semantic subtyping relation by adopting a set-theoretic interpretation in the manner of [Frisch et al., 2008], in contrast with more traditional subtyping through direct syntactic rules. As a main contribution, we show how to decide this relation.

This work is motivated by a growing need for polymorphic type systems for programming languages that manipulate XML data. For instance, XQuery [Boag et al., 2006] is the standard query and functional language designed for querying collections of XML data. The support of higher-order functions, currently missing from XQuery, appears in the requirements for the forthcoming XQuery 3.0 language [Engovatov & Robie, 2010]. This results in an increasing demand in algorithms for proving or disproving statements such as the one

of the example (6.4) with polymorphic types, but also with types of higher-order functions (like the traditional `map` and `fold` functions), or more generally, statements involving the subtyping relation over a type algebra with recursive, product, function, intersection, union, and complement types together with type variables and universal quantification over them.

6.1.2 Semantic Subtyping with Logical Solvers

During the last few years, a growing interest has been seen in the use of logical solvers such as satisfiability solver and satisfiability-modulo solvers in the context of functional programming and static type checking [Bierman et al., 2010; Benedikt & Cheney, 2010]. In particular, solvers for tree logics [Genevès et al., 2007b; de Moura & Bjørner, 2008] are used as basic building blocks for type systems for XQuery.

The main idea in this chapter is a type-checking algorithm for polymorphic types based on deciding subtyping through a logical solver. To decide whether τ is a subtype of type τ', we first construct equivalent logical formulas φ_τ and $\varphi_{\tau'}$ and then check the validity of the formula $\psi = \varphi_\tau \Rightarrow \varphi_{\tau'}$ by testing the unsatisfiability of $\neg\psi$ using the satisfiability-testing solver. This technique corresponds to semantic subtyping [Frisch et al., 2008] since the underlying logic is inherently tied to a set-theoretic interpretation. Semantic subtyping has been applied to a wide variety of types including refinement types [Bierman et al., 2010] and types for XML such as regular tree types [Hosoya et al., 2005b], function types [Benzaken et al., 2003], and XPath [Clark & DeRose, 1999] expressions [Genevès et al., 2007b].

This fruitful connection between logics, their decision procedures, and programming languages permitted to equip the latter with rich type systems for sophisticated programming constructs such as expressive pattern-matching and querying techniques. The potential benefits of this interconnection crucially depend on the expressivity of the underlying logics. Therefore, there is an increasing demand for more and more expressiveness. For example, in the context of XML:

- SMT solvers like [de Moura & Bjørner, 2008] offer an expressive power that corresponds to a fragment of first-order logic in order to solve the intersection problem between two queries [Benedikt & Cheney, 2010];
- full first-order logic solvers over finite trees [Genevès et al., 2007b] solve containment and equivalence of XPath expressions;
- monadic second-order logic solvers over trees, and – equivalent yet much more effective – satisfiability-solvers for μ-calculus over trees [Genevès et al., 2007b] are used to solve query containment problems in the presence of type constraints.

6.1.3 Contributions

To the best of our knowledge, novelty of our work is threefold. It is the first work that:

- proves the decidability of semantic subtyping for polymorphic types with function, product, intersection, union, and complement types, as defined by Castagna and Xu [Castagna & Xu, 2011], and gives a precise complexity upper-bound: $2^{(n)}$, where n is the size of types being checked. Decidability was only conjectured by Castagna and Xu before our result, although they have now proved it independently; our result on complexity is still the only one. In addition, we provide an effective implementation of the decision procedure.

- produces counterexamples whenever subtyping does not hold. These counterexamples are valuable for programmers as they represent evidence that the relation does not hold.

- pushes the integration between programming languages and logical solvers to a very high level. The logic in use is not only capable to range over higher order functions, but it is also capable of expressing values from semantic domains that correspond to monadic second-order logic such as XML tree types [Genevès et al., 2007b]. This shows that such solvers can become the core of XML-centric functional languages type-checkers such as those used in CDuce [Benzaken et al., 2003] or XDuce [Hosoya & Pierce, 2003].

6.1.4 Structure of the Chapter

We introduce the semantic subtyping framework in Section 6.2 where we start with the monomorphic type algebra (without type variables). We present the tree logic in which we model semantic subtyping in Section 6.3. We detail the logical encoding of types in Section 6.4. Then, in Section 6.5 we extend the type algebra with type variables, and state the main result of the chapter: we show how to decide the subtyping relation for the polymorphic case in exponential time. We report on practical experiments using the implementation in Section 6.6. Finally, we discuss related work in Section 6.7 before concluding in Section 6.8.

6.2 Semantic Subtyping Framework

In this section, we present the type algebra we consider: we introduce its syntax and define its semantics in terms of semantic domains. This framework is the one described at length in [Frisch et al., 2008]; we do not discuss its properties here but just give the necessary definitions, that we will then extend with type variables in Section 6.5.

6.2.1 Types

Type terms are defined using the following grammar:

τ	::=		
		b	basic type
	$\mid$	$\tau \times \tau$	product type
	$\mid$	$\tau \to \tau$	function type
	$\mid$	$\tau \vee \tau$	union type
	$\mid$	$\neg\tau$	complement type
	$\mid$	$\mathbf{0}$	empty type
	$\mid$	v	recursion variable
	$\mid$	$\mu v.\tau$	recursive type

We consider μ as a binder and define the notions of free and bound variables and closed terms as standard. A type is a closed type term which is well-formed in the sense that:

- the negation operator only occurs in front of closed types;
- every occurrence of a recursion variable is separated from its binder by at least one occurrence of the product or arrow constructor.

So, for example, $\mu v.\mathbf{0} \vee v$ is not well-formed, nor is $\mu v.\mathbf{0} \to \neg v$.

Additionally, the following abbreviations are defined:

$$\tau_1 \wedge \tau_2 \stackrel{\text{def}}{=} \neg(\neg\tau_1 \vee \neg\tau_2)$$

and

$$\mathbf{1} = \neg\mathbf{0}$$

6.2.2 Semantic domain

Consider an arbitrary set $\mathcal{C}$ of constants. From it, we define the semantic domain $\mathcal{D}$ as the set of ds generated by the following grammar, where c ranges over constants in $\mathcal{C}$:

d	::=		domain element
		c	base constant
	$\mid$	(d, d)	pair
	$\mid$	$\{(d, d'), \ldots, (d, d')\}$	function
d'	::=		extended domain element
		d	
	$\mid$	Ω	error

The function terms are finite sets of pairs representing nondeterministic partial functions from $\mathcal{D}$ to $\mathcal{D} \cup \{\Omega\}$: each pair (d, d') in the set means that, when given d as an argument, the function may yield d' as a result. If d does not appear as the first element of any pair, the operational interpretation is that the function can still accept d as an argument but will not yield a result:

this represents a computation which does not terminate. A pair of the form (d, Ω) is used to represent a function rejecting d as an argument: when given d, it yields an error.

This grammar is only able to represent functions which diverge but on a finite number of possible arguments. However it is shown in [Frisch et al., 2008] (Lemma 6.32) that considering only those functions does not affect the subtyping relation.

6.2.3 Interpretation

We suppose we have an interpretation $\mathbb{B}[\![\cdot]\!]$ of basic types b as subsets of $\mathcal{C}$.

The predicate $(d' : \tau)$ where d' is an element of $\mathcal{D}$ or Ω and τ is a type is defined recursively in the following way:

$$\begin{aligned}
(\Omega : \tau) &= \mathsf{false} \\
(c : b) &= c \in \mathbb{B}[\![b]\!] \\
((d_1, d_2) : \tau_1 \times \tau_2) &= (d_1 : \tau_1) \wedge (d_2 : \tau_2) \\
(\{(d_1, d'_1), \ldots, (d_n, d'_n)\} : \tau_1 \to \tau_2) &= \forall i, (d_i : \tau_1) \Rightarrow (d'_i : \tau_2) \\
(d : \tau_1 \vee \tau_2) &= (d : \tau_1) \vee (d : \tau_2) \\
(d : \neg\tau) &= \neg(d : \tau) \\
(d : \mu v.\tau) &= (d : \tau\{^{\mu v.\tau}/_{v}\}) \\
(d : \tau) &= \mathsf{false} \text{ in any other case}
\end{aligned}$$

To prove this definition is well-founded, we first define the shallow depth of a type term as the longest path, in its syntactic tree, starting from the root and consisting only of μ, $\vee$, and $\neg$ nodes. We then use the following ordering on pairs (d', t) :

- $d'_1 \leqslant d'_2$ if d'_1 is a subterm of d'_2
- $\tau_1 \leqslant \tau_2$ if the shallow depth of τ_1 is less than the shallow depth of τ_2
- pairs are ordered lexicographically, i. e. $(d'_1, \tau_1) \leqslant (d'_2, \tau_2)$ if either $d'_1 < d'_2$ or $d'_1 = d'_2$ and $\tau_1 \leqslant \tau_2$.

Now we can see that all occurrences of the predicate on the right-hand side of the definition are for pairs strictly smaller than the one on the left (in the case of $\mu v.\tau$, this is due to the well-formedness constraint: the variable being substituted can only appear below a $\times$ or $\to$ node). Because all terms and types are finite, this makes the definition well-founded.

The interpretation of types as parts of $\mathcal{D}$ is then defined as $[\![\tau]\!] = \{d \mid (d : \tau)\}$. Note that Ω is not part of any type, as expected.

In this framework, we consider XML types as regular tree languages. An XML tree type is interpreted as the set of documents that match the type.

6.2.4 Subtyping

The subtyping relation is defined as $\tau_1 \leqslant \tau_2 \Leftrightarrow [\![\tau_1]\!] \subset [\![\tau_2]\!]$, or, equivalently, $[\![\tau_1 \wedge \neg\tau_2]\!] = \emptyset$.

6.3 Tree logic framework

In this section we introduce the logic in which we model the semantic subtyping framework. This logic is a subset of the one proposed in [Genevès et al., 2007b]: a variant of μ-calculus whose models are finite trees. We first introduce below the syntax and semantics of the logic, before tuning it for representing types.

6.3.1 Formulas

Formulas are defined thus:

φ, ψ	$::=$		formula
		$\top$	true
	$\mid$	$p \mid \neg p$	atomic proposition (negated)
	$\mid$	X	variable
	$\mid$	$\varphi \vee \psi$	disjunction
	$\mid$	$\varphi \wedge \psi$	conjunction
	$\mid$	$\langle a \rangle \varphi \mid \neg \langle a \rangle \top$	existential (negated)
	$\mid$	$\mu(X_i = \varphi_i)_{i \in I}$ in ψ	(least) n-ary fixpoint

where $a \in \{1, 2\}$ are programs, and I is a finite set. Atomic propositions p correspond to labels from a countable set Σ. Additionally, we use the abbreviation $\mu X.\varphi$ for $\mu(X = \varphi)$ in φ.

6.3.2 Semantic domain

The semantic domain is the set $\mathcal{F}$ of focused trees defined by the following syntax, where we have an alphabet Σ of labels, ranged over by σ:

t	$::=$	$\sigma[tl]$	tree
tl	$::=$		list of trees
		ϵ	empty list
	$\mid$	$t :: tl$	cons cell
c	$::=$		context
		(tl, Top, tl)	root of the tree
	$\mid$	$(tl, c[\sigma], tl)$	context node
f	$::=$	(t, c)	focused tree

A focused tree (t, c) is a pair consisting of a tree t and its context c. The context $(tl, c[\sigma], tl)$ comprises three components: a list of trees at the left of the current tree in reverse order (the first element of the list is the tree immediately to the left of the current tree), the context above the tree, and a list of trees at the right of the current tree. The context above the tree may be *Top* if the current tree is at the root, otherwise it is of the form $c[\sigma]$ where σ is the label of the enclosing element and c is the context in which the enclosing element occurs.

The name of a focused tree is defined as $\mathtt{nm}(\sigma[tl], c) = \sigma$.

We now describe how to navigate focused trees, in binary style. There are four directions, or modalities, that can be followed: for a focused tree f, $f\langle 1 \rangle$ changes the focus to the first child of the current tree, $f\langle 2 \rangle$ changes the focus to the next sibling of the current tree, $f\langle \overline{1} \rangle$ changes the focus to the parent of

the tree if the current tree is a leftmost sibling, and $f\,\langle\overline{2}\rangle$ changes the focus to the previous sibling.

Formally, we have:

$$(\sigma[t :: tl], c)\,\langle 1\rangle \stackrel{\text{def}}{=} (t, (\epsilon, c[\sigma], tl))$$
$$(t, (tl_l, c[\sigma], t' :: tl_r))\,\langle 2\rangle \stackrel{\text{def}}{=} (t', (t :: tl_l, c[\sigma], tl_r))$$
$$(t, (\epsilon, c[\sigma], tl))\,\langle\overline{1}\rangle \stackrel{\text{def}}{=} (\sigma[t :: tl], c)$$
$$(t', (t :: tl_l, c[\sigma], tl_r))\,\langle\overline{2}\rangle \stackrel{\text{def}}{=} (t, (tl_l, c[\sigma], t' :: tl_r))$$

When the focused tree does not have the required shape, these operations are not defined.

6.3.3 Interpretation

Formulas are interpreted as subsets of $\mathcal{F}$ in the following way, where V is a mapping from variables to formulas:

$$[\![\top]\!]_V \stackrel{\text{def}}{=} \mathcal{F} \qquad [\![p]\!]_V \stackrel{\text{def}}{=} \{f \mid \mathtt{nm}(f) = p\}$$
$$[\![X]\!]_V \stackrel{\text{def}}{=} V(X) \qquad [\![\neg p]\!]_V \stackrel{\text{def}}{=} \{f \mid \mathtt{nm}(f) \neq p\}$$
$$[\![\varphi \vee \psi]\!]_V \stackrel{\text{def}}{=} [\![\varphi]\!]_V \cup [\![\psi]\!]_V \qquad [\![\varphi \wedge \psi]\!]_V \stackrel{\text{def}}{=} [\![\varphi]\!]_V \cap [\![\psi]\!]_V$$

$$[\![\langle a\rangle\,\varphi]\!]_V \stackrel{\text{def}}{=} \{f\,\langle\overline{a}\rangle \mid f \in [\![\varphi]\!]_V \wedge f\,\langle\overline{a}\rangle \text{ defined}\}$$
$$[\![\neg\,\langle a\rangle\,\top]\!]_V \stackrel{\text{def}}{=} \{f \mid f\,\langle a\rangle \text{ undefined}\}$$

$$[\![\mu(X_i = \varphi_i)_{i\in I} \text{ in } \psi]\!]_V \stackrel{\text{def}}{=}$$
$$\text{let } S = \{(T_i) \in \mathcal{P}(\mathcal{F})^I \mid \forall j \in I, [\![\varphi_j]\!]_{V[\overline{T_i/X_i}]} \subset T_j\} \quad \text{in}$$
$$\text{let } (U_j) = \left(\textstyle\bigcap_{(T_i)\in S} T_j\right)_{j\in I} \quad \text{in} \quad [\![\psi]\!]_{V[\overline{U_i/X_i}]}$$

where $V[\overline{T_i/X_i}](X) = V(X)$ if $X \notin \{X_i\}$ and T_i if $X = X_i$.

The lemma 4.2 of [Genevès et al., 2007b] says that the interpretation of a fixpoint formula is equal to the union of the interpretations of all its finite unfoldings (where unfolding is defined as usual). A consequence (detailed in [Genevès et al., 2007b]) is that the logic is closed under negation, i. e. for any closed φ, $\neg\varphi$ can be expressed in the syntax using De Morgan's relations and this definition:

$$\neg\,\langle a\rangle\,\varphi \stackrel{\text{def}}{=} \neg\,\langle a\rangle\,\top \vee \langle a\rangle\,\neg\varphi$$
$$\neg\mu(X_i = \varphi_i) \text{ in } \psi \stackrel{\text{def}}{=} \mu(X_i = \neg\varphi_i\{\overline{X_i/\neg X_i}\}) \text{ in } \neg\psi\{\overline{X_i/\neg X_i}\}$$

In the following, we consider only closed formulas and write $[\![\varphi]\!]$ for $[\![\varphi]\!]_\emptyset$.

6.4 Logical Encoding

In the context of the present chapter, we want finite tree models of the logic to correspond to types introduced in section 6.2. Thus, we first extend the alphabet of node labels to be able to reason with type constructors. Then, we present the translation of a type into a logical formula.

6.4.1 Representation of domain elements

Let $\mathcal{T}$ be the set of (unfocused) trees. Set $\mathcal{C} = \{\mathbb{B}[tl] \mid tl \in \mathcal{T}^*\}$, where $\mathbb{B}$ is a label **not in** Σ : the set of trees with a distinguished root $\mathbb{B}$. Let $\mathcal{T}_{ext}$ be the set of trees obtained by extending Σ with the four extra labels $(\rightarrow), (\times), \mathbb{B}$ and Ω. Then $\mathcal{D}_\Omega$ can straightforwardly be embedded into $\mathcal{T}_{ext}$ in the following way:

$$\begin{aligned}
\mathsf{tree}(c) &= c \\
\mathsf{tree}(\Omega) &= \Omega[\epsilon] \\
\mathsf{tree}(d, d') &= (\times)[\mathsf{tree}(d) :: \mathsf{tree}(d') :: \epsilon]
\end{aligned}$$

$$\mathsf{tree}(\{(d_1, d'_1), \ldots, (d_n, d'_n)\}) = (\rightarrow)[\mathsf{tree}(d_1, d'_1) :: \ldots :: \mathsf{tree}(d_n, d'_n) :: \epsilon]$$

In the following we consider this embedding implicitly done, so $\mathcal{D}_\Omega \subset \mathcal{T}_{ext}$.

6.4.2 Translation of types

First of all, we can define basic types b, which are to represent sets of trees with no special nodes but the distinguished root $\mathbb{B}$, as the (closed) base formulas of the logic. The full interpretation of formulas uses sets of focused trees, but note that a toplevel formula cannot contain any constraint on what is above or to the left of the node at focus, so it can be considered as describing just a list of trees. The interpretation of a base type will then be a $\mathbb{B}$ root whose list of children is described by the formula. Formally:

$$\mathbb{B}[\![\varphi]\!] \stackrel{\text{def}}{=} \{\mathbb{B}[t :: tl_2] \mid (t, (tl_1, c[\sigma], tl_2)) \in [\![\varphi]\!]\}$$

Note how the only part of the context taken into account in defining the semantics is the list of following siblings of the current node.

Then, we translate the types into extended formulas obtained (as for extended trees) by adding to Σ the labels $(\times), (\rightarrow), \Omega$ and $\mathbb{B}$. Straightforwardly these formulas denote lists of trees in $\mathcal{T}_{ext}$.

First define the following formulas:

$$\begin{aligned}
\mathsf{isbase} &= \mu X.((\neg \langle 1\rangle \top \vee \langle 1\rangle X) \wedge (\neg \langle 2\rangle \top \vee \langle 2\rangle X) \\
&\qquad \wedge \neg\mathbb{B} \wedge \neg(\rightarrow) \wedge \neg(\times) \wedge \neg\Omega) \\
\mathsf{error} &= \Omega \wedge \neg \langle 1\rangle \top \\
\mathsf{isd} &= \mu X.(\\
&\qquad (\mathbb{B} \wedge \langle 1\rangle \mathsf{isbase}) \vee \\
&\qquad ((\times) \wedge \langle 1\rangle (X \wedge \langle 2\rangle (X \wedge \neg \langle 2\rangle \top))) \vee \\
&\qquad ((\rightarrow) \wedge (\neg \langle 1\rangle \top \vee \\
&\qquad\quad \langle 1\rangle \mu Y.((\neg \langle 2\rangle \top \vee \langle 2\rangle Y) \wedge \\
&\qquad\qquad (\times) \wedge \langle 1\rangle (X \wedge \langle 2\rangle ((X \vee \mathsf{error}) \wedge \neg \langle 2\rangle \top)) \\
&\qquad))))
\end{aligned}$$

isbase selects all tree lists which do not contain any of the special labels (the fixpoint is for selecting all the nodes). error is straightforward. isd selects all elements of $\mathcal{D}$ (actually, all tree lists whose first element is in $\mathcal{D}$): either they are a constant (a $\mathbb{B}$ node with a base list as children), or a pair (a $(\times)$ node with exactly two children each of which is itself in $\mathcal{D}$), or a function: a $(\to)$ node with either no children at all or a list of children (described by Y) all of which are pairs whose second element may be error.

We now associate to every type τ the formula $\mathsf{fullform}(\tau) = \mathsf{isd} \wedge \mathsf{form}(\tau)$, with $\mathsf{form}(\tau)$ defined as follows, where X_v is a different variable for every v and is also different from X:

$$
\begin{aligned}
\mathsf{form}(b) &= \mathbb{B} \wedge \langle 1\rangle\, b \\
\mathsf{form}(\tau_1 \times \tau_2) &= (\times) \wedge \langle 1\rangle\, (\mathsf{form}(\tau_1) \wedge \langle 2\rangle\, \mathsf{form}(\tau_2)) \\
\mathsf{form}(\tau_1 \to \tau_2) &= (\to) \wedge (\neg \langle 1\rangle\, \top \vee \\
&\qquad \langle 1\rangle\, \mu X.((\neg \langle 2\rangle\, \top \vee \langle 2\rangle\, X) \\
&\qquad\quad \wedge \langle 1\rangle\, (\neg \mathsf{form}(\tau_1) \vee \langle 2\rangle\, \mathsf{form}(\tau_2))) \\
&\qquad)\\
\mathsf{form}(\tau_1 \vee \tau_2) &= \mathsf{form}(\tau_1) \vee \mathsf{form}(\tau_2) \\
\mathsf{form}(\neg\tau) &= \neg \mathsf{form}(\tau) \\
\mathsf{form}(\mathbf{0}) &= \neg\top \\
\mathsf{form}(v) &= X_v \\
\mathsf{form}(\mu v.\tau) &= \mu X_v.\mathsf{form}(\tau)
\end{aligned}
$$

Recall that basic types b are themselves formulas, but that their interpretation as a type is different from their interpretation as a formula (see the first paragraph of Section 6.4.2 and the definition of $\mathbb{B}[\![\varphi]\!]$, the interpretation as a type, in terms of $[\![\varphi]\!]$, the interpretation as a formula). This explains why the translation of b contains b itself. The translation of product types is simple: it describes a $(\times)$ node whose first child is described by $\mathsf{form}(\tau_1)$ and has a following sibling described by $\mathsf{form}(\tau_2)$. The translation of arrow types has a structure similar to what appeared in isd: it describes a $(\to)$ node with either no children or a list of children recursively described by X (each node has either no following sibling or a following sibling itself described by X). Each of these nodes must have a first child which either is not of type τ_1 or has a next sibling of type τ_2 — this means that these nodes represent pairs (d_i, d'_i) such that $(d_i : \tau_1) \Rightarrow (d'_i : \tau_2)$. The attentive reader may notice that the formula $\mathsf{form}(\tau_1 \to \tau_2)$ does not enforce in itself that all children of the $(\to)$ node are actually pairs; the reason for that is that isd already enforces it.

We can see that the formulas in the translation do not contain any $\langle 2\rangle$ at toplevel (i. e. not under $\langle 1\rangle$), nor does isd. This means they describe a single tree (they say nothing on its siblings), or in other words that in their interpretation as focused trees, the context is completely arbitrary, as it is not constrained in any way. Formally, we thus define the restricted interpretation

of extended formulas as follows:

$$\mathbb{F}[\![\varphi]\!] \stackrel{\text{def}}{=} \{t \mid (t, c) \in [\![\varphi]\!]\}$$

That is, we drop the context completely.

Then we have $\mathbb{F}[\![\mathsf{fullform}(\tau)]\!] = [\![\tau]\!]$. This is a particular case of the property for polymorphic types which will be proved in the following section.

The main consequence of this property is that a type τ is empty if and only if the interpretation of the corresponding formula is empty — which is equivalent to the formula being unsatisfiable. Because their exists a satisfiability-checking algorithm for this tree logic [Genevès et al., 2007b], this means this translation gives an alternative way to decide the classical semantic subtyping relation as defined in [Frisch et al., 2008]. More interestingly, it yields a decision procedure for the subtyping relation in the polymorphic case as well, as we will explain in the next section.

6.5 Polymorphism: Supporting Type Variables

So far we have described a new, logic-based approach to a question — semantic subtyping in the presence of intersection, negation and arrow types — which had already been studied. We now show how this new approach allows us, in a very natural way, to encompass the latest work by adding polymorphism to the types along the lines of [Castagna & Xu, 2011].

We add to the syntax of types *variables*, α, β, γ taken from a countable set $\mathcal{V}$. If τ is a polymorphic type, we write $\mathsf{var}(\tau)$ the set of variables it contains and call ground type a type with no variable. We sometimes write $\tau(\overline{\alpha})$ to indicate that $\mathsf{var}(\tau)$ is included in $\overline{\alpha}$.

6.5.1 Polymorphic Subtyping: a problem of definition

The intuition of subtyping in the presence of type variables is that $\tau_1(\overline{\alpha}) \leqslant \tau_2(\overline{\alpha})$ should hold true whenever, independently of the variables $\overline{\alpha}$, any value of type τ_1 has type τ_2 as well. However the correct definition of 'independently' is not obvious. It should look like this:

$$\forall \overline{\alpha}, [\![\tau_1(\overline{\alpha})]\!] \subset [\![\tau_2(\overline{\alpha})]\!]$$

but because variables are abstractions, it is not completely clear over what to quantify them. As mentioned in [Hosoya et al., 2009], a candidate — naive — definition would use ground substitutions, that is, if the inclusion of interpretations always holds when variables are replaced with ground types, then the subtyping relation holds:

$$\tau_1(\overline{\alpha}) \leqslant \tau_2(\overline{\alpha}) \Leftrightarrow \forall \overline{\tau} \text{ ground types}, [\![\tau_1(\overline{\tau}/\overline{\alpha})]\!] \subset [\![\tau_2(\overline{\tau}/\overline{\alpha})]\!] \tag{6.5}$$

Obviously the condition on the right must be necessary for subtyping to hold. But deciding that it is sufficient as well makes the relation unsatisfactory and somehow counterintuitive, as remarked in [Hosoya et al., 2009]. Indeed, suppose

int is an indivisible type, that is, that it has no subtype beside $\mathbf{0}$ and itself. Then the following would hold:

$$\mathsf{int} \times \alpha \leqslant (\mathsf{int} \times \neg\mathsf{int}) \vee (\alpha \times \mathsf{int}) \tag{6.6}$$

This relation abuses the definition by taking advantage of the fact that for any ground type τ, either $[\![\mathsf{int}]\!] \subset [\![\tau]\!]$ or $[\![\tau]\!] \subset [\![\neg\mathsf{int}]\!]$. In the first case, because $[\![\tau]\!] \subset ([\![\neg\mathsf{int}]\!] \cup [\![\mathsf{int}]\!])$, we have $[\![\mathsf{int} \times \tau]\!] \subset [\![\mathsf{int} \times \neg\mathsf{int}]\!] \cup [\![\mathsf{int} \times \mathsf{int}]\!]$ and then the second member of the union is included in $[\![\tau \times \mathsf{int}]\!]$. In the second case, we directly have $[\![\mathsf{int} \times \tau]\!] \subset [\![\mathsf{int} \times \neg\mathsf{int}]\!]$.

This trick, which only works with indivisible ground types, not only shows that candidate definition (6.5) yields bizarre relations where a variable occurs in unrelated positions on both sides. It also means the candidate definition is very sensitive to the precise semantics of base types, since it distinguishes indivisible types from others. More precisely, it means that refining the collection of base types, for example by adding types even and odd, can break subtyping relations which held true without these new types — this is simply due to the fact that it increases the set over which $\overline{\tau}$ is quantified in (6.5), making the relation stricter. This could hardly be considered a nice feature of the subtyping relation.

The conclusion is thus that the types in (6.6) should be considered related by chance rather than by necessity, hence not in the subtyping relation, and that quantifying over all possible ground types is not enough; in other words, candidate definition (6.5) is too weak and does not properly reflect the intuition of 'independently of the variables'. Indeed, (6.6) is in fact dependent on the variable as we saw, the point being that there are only two cases and that the convoluted right-hand type is crafted so that the relation holds in both of them, though for different reasons.

In order to restrict the definition of subtyping, [Hosoya et al., 2009], which concentrates on XML types, uses a notion of marking: some parts of a value can be marked (using paths) as corresponding to a variable, and the relation 'a value has a type' is changed into 'a marked value matches a type', so the semantics of a type is not a set of values but of pairs of a value and a marking. This is designed so that it integrates well in the XDuce language, which has pattern-matching but no higher-order functions (hence no arrow types), so their system is tied to the operational semantics of matching and provides only a partial solution.

The question of finding the correct definition of semantic subtyping in the polymorphic case was finally settled very recently by Castagna and Xu [Castagna & Xu, 2011]. Their definition does, in the same way as (6.5), follow the idea of a universal quantification over possible meanings of variables but solves the problem raised by (6.6) by using a much larger set of possible meanings — thus yielding a stricter relation. More precisely, variables are allowed to represent not just ground types but any arbitrary part of the semantic domain; furthermore, the semantic domain itself must be large enough, which is embodied by the notion of convexity. We refer the reader to [Castagna & Xu, 2011] for a detailed discussion of this property and its relation to the notion

of parametricity studied by Reynolds in [Reynolds, 1983]; we will here limit ourselves to introducing the definitions strictly necessary for the discussion at hand.

In this work, we do not use this definition with its universal quantification directly. Rather, we retain from [Hosoya et al., 2009] the idea of tagging (pieces of) values which correspond to variables, but do so in a more abstract way, by extending the semantic domain, and define a fixed interpretation of polymorphic types in this extended domain as a straightforward extension of the monomorphic framework. We then show how to build a set-theoretic model of polymorphic types, in the sense of [Castagna & Xu, 2011], based on this domain, and prove that the inclusion relation on fixed interpretations is equivalent to the full subtyping relation induced by this model. Finally, we explain briefly the notion of convexity and show that this model is convex, implying that this relation is, in fact, the semantic subtyping relation on polymorphic types, as defined in [Castagna & Xu, 2011]. These steps are formally detailed in the following section.

6.5.2 Interpretation of polymorphic types

Let Λ be an infinite set of optional labels, and ι an injective function from $\mathcal{V}$ to Λ. (It would be possible to set $\Lambda = \mathcal{V}$, but for clarity we prefer to distinguish labels which tag elements of the semantic domain from variables which occur in types.) We extend the grammar of (extended) trees by allowing any node to bear, in addition to its single σ label from $\Sigma \cup \{(\to), (\times), \mathbb{B}, \Omega\}$, any (finite) number of labels from Λ. We write it $\sigma_L[tl]$ where L is a finite part of Λ. We extend $\mathcal{C}$ and $\mathcal{D}$ accordingly. When using the non-tree form of types, for instance (d_1, d_2), we indicate the set of root labels on the bottom right like this: $(d_1, d_2)_L$ (here L is the set of labels borne by the $(\times)$ node constituting the root of the pair tree).

We then extend the predicate defining the interpretation of types given in Section 6.2.3 with the following additional case:

$$(\sigma_L[tl] : \alpha) = \iota(\alpha) \in L$$

In other words, the interpretation of a type variable is the set of all trees whose root bears the label corresponding to that variable. The other cases are unchanged, except that the semantic domain is now much larger. This means that the same definition leads to larger interpretations; in particular, the interpretation of a (nonempty) ground type is always an infinite set which contains all possible labellings for each of its trees.

Subtyping over polymorphic types is then defined, as before, as set inclusion between interpretations:

$$\tau_1(\overline{\alpha}) \leqslant \tau_2(\overline{\alpha}) \Leftrightarrow [\![\tau_1(\overline{\alpha})]\!] \subset [\![\tau_2(\overline{\alpha})]\!] \tag{6.7}$$

It may seem strange to give type variables a fixed interpretation, and on the other hand it may seem surprising that this definition of subtyping does not actually contain any quantification and is nevertheless stronger than (6.5)

which contains one. The keypoint is that a form of universal quantification is implicit in the extension of the semantic domain: in some sense, the interpretation of a variable represents all possible values of the variable at once. Indeed, for any variable α and any tree d in the domain, there always exist both an infinity of copies of d which are in the interpretation of α and another infinity of copies which are not. From the point of view of logical satisfiability, this makes the domain big enough to contain all possible cases.

In order to show that, despite the appearances, Definition (6.7) accurately represents a relation that holds independently of the variables, we rely, as discussed above, on the formal framework developed by Castagna and Xu [Castagna & Xu, 2011]. For this, we first introduce assignments η: functions from $\mathcal{V}$ to $\mathcal{P}(\mathcal{D})$ (where $\mathcal{D}$ is the extended semantic domain with labels). Thus an assignment attributes to each variable an arbitrary set of elements from the semantic domain.

We then define the interpretation of a type relative to an assignment in the following way: the predicate $(d' :_\eta \tau)$ is defined inductively in the same way as the $(d' : \tau)$ of Section 6.2.3 but with the additional clause:

$$(d :_\eta \alpha) = d \in \eta(\alpha).$$

The interpretation of the polymorphic type τ relative to the assignment η is then $[\![\tau]\!]\eta = \{d \mid (d :_\eta \tau)\}$. This defines an infinity of possible interpretations for a type, depending on the actual values assigned to the variables, and constitutes a set-theoretic model of types in the sense of [Castagna & Xu, 2011]. The subtyping relation induced by this model is the following:

$$\tau_1(\overline{\alpha}) \leqslant \tau_2(\overline{\alpha}) \Leftrightarrow \forall \eta \in \mathcal{P}(\mathcal{D})^{\mathcal{V}}, [\![\tau_1(\overline{\alpha})]\!]\eta \subset [\![\tau_2(\overline{\alpha})]\!]\eta \tag{6.8}$$

which we can more easily compare to the candidate definition (6.5): it does in the same way quantify over possible meanings of the variables but uses a much larger set of possible meanings, yielding a stricter relation. We will now prove that this relation is, for our particular model, actually equivalent to (6.7).

For this, let us first define the canonical assignment η_ι as follows:

$$\eta_\iota(\alpha) \stackrel{\text{def}}{=} \{\sigma_L[tl] \in \mathcal{D} \mid \iota(\alpha) \in L\}.$$

Then it is easily seen that the fixed interpretation $[\![\tau]\!]$ of a polymorphic type is the same as its interpretation relative to the canonical assignment, $[\![\tau]\!]\eta_\iota$. What we would like to prove is that the canonical assignment is somehow representative of all possible assignments, making the fixed interpretation sufficient for the purpose of defining subtyping. This is done by the following lemma and corollary.

Lemma 6.5.1. *Let V be a finite part of $\mathcal{V}$. Let η be an assignment. Let T be the set of all types τ such that $\mathsf{var}(\tau) \subset V$. Then there exists a function $F_V^\eta : \mathcal{D} \to \mathcal{D}$ such that: $\forall \tau \in T, \forall d \in \mathcal{D}, d \in [\![\tau]\!]\eta \Leftrightarrow F_V^\eta(d) \in [\![\tau]\!]\eta_\iota$.*

Proof. For d in $\mathcal{D}$, let $L(d) = \{\iota(\alpha) \mid \alpha \in V \wedge d \in \eta(\alpha)\}$. Since V is finite, $L(d)$ is finite as well. We define $F_V^\eta(d)$ inductively as follows:

- if $d = \mathbb{B}_L[tl]$ then $F_V^\eta(d) = \mathbb{B}_{L(d)}[tl]$
- if $d = (d_1, d_2)_L$ then $F_V^\eta(d) = (F_V^\eta(d_1), F_V^\eta(d_2))_{L(d)}$
- $F_V^\eta(\Omega) = \Omega$
- if $d = \{(d_1, d_1'), \ldots, (d_n, d_n')\}_L$ then
  $F_V^\eta(d) = \{(F_V^\eta(d_1), F_V^\eta(d_1')), \ldots, (F_V^\eta(d_n), F_V^\eta(d_n'))\}_{L(d)}$

So F_V^η preserves the structure but changes the labels so that the root node of $F_V^\eta(d)$ is labelled with $L(d)$ and so on inductively for its subterms.

Let $\mathcal{P}(d, \tau) = d \in [\![\tau]\!]\eta \Leftrightarrow F_V^\eta(d) \in [\![\tau]\!]\eta_\iota$. We prove that it holds for all pairs (d, τ) such that τ is in T by induction on those pairs, using the ordering relation on them defined in Section 6.2.3, noticing that $\tau \in T$ implies that all subterms (and unfoldings) of τ are in T as well. The base cases are:

- if τ is a variable. Then it is in V by hypothesis and $\mathcal{P}(d, \tau)$ is true by definition of $L(d)$.
- if it is a base type. Then $\mathcal{P}(d, \tau)$ is true because the interpretation of τ is independent of assignments and labellings.

For the inductive cases, we suppose the property true for all strictly smaller pairs (d, τ) such that τ is in T.

- For the arrow and product cases, the inductive definition of F_V^η makes the result straightforward.
- For the negation and disjunction cases, the result is immediate from the induction hypothesis.
- For $\mu v.\tau$, recall that the well-formedness constraint on types implies that the type's unfolding has a strictly smaller shallow depth than the original type, hence we can use the induction hypothesis on the unfolding and conclude.

□

Corollary 6.5.2. *Let τ be a type.* $\bigcup_{\eta \in \mathcal{P}(\mathcal{D})^\mathcal{V}} [\![\tau]\!]\eta = \emptyset$ *if and only if* $[\![\tau]\!]\eta_\iota = \emptyset$.

Proof. If the union is not empty, there exists η and d such that $d \in [\![\tau]\!]\eta$. From the previous lemma we then have $F^\eta_{\mathsf{var}(\tau)}(d) \in [\![\tau]\!]\eta_\iota$. □

This corollary shows that the canonical assignment is representative of all possible assignments and implies that the subtyping relation defined by (6.7) is equivalent to the one defined by (6.8).

Convexity of the model.. Definition (6.8) corresponds to semantic subtyping as defined in [Castagna & Xu, 2011], but only on the condition that the underlying model of types be convex. Indeed, we can see that this definition is dependent on the set of possible assignments, which itself depends on the chosen (abstract) semantic domain, so it is reasonable to think that increasing the semantic domain could restrict the relation further. In other words, for the definition to be correct, the domain must be large enough to cover all cases. Castagna and Xu's convexity characterises this notion of 'large enough'. The property is the following: a set-theoretic model of types is convex if, whenever a finite collection of types τ_1 to τ_n each possess a nonempty interpretation relative to some assignment, then there exists a common assignment making all interpretations nonempty at once. This reflects the idea that there are enough elements in the domain to witness all the cases.

In our case, it comes as no surprise that the extended model of types is convex since any nonempty ground type has an infinite interpretation, which, as proved in [Castagna & Xu, 2011], is a sufficient condition. But we need not even rely on this result since Corollary 6.5.2 proves a property even stronger than convexity: having a nonempty interpretation relative to some assignment is the same as having a nonempty interpretation relative to the common canonical assignment. This stronger property makes the apparently weaker relation defined by (6.7) equivalent, in our particular model, to the full semantic subtyping relation Castagna and Xu defined. This allows us to reduce the problem of deciding their relation to a question of inclusion between fixed interpretations, making the addition of polymorphism a mostly straightforward extension to the logical encoding we presented for the monomorphic case.

Interestingly, in [Castagna & Xu, 2011] the authors suggest that convexity constrains the relation enough that it should allow reasoning on types, similarly to the way parametricity allowed Wadler [Wadler, 1989] to deduce 'theorems for free' from typing information. The fact that our logical reasoning approach very naturally has this convexity property — indeed, it is difficult to think of a logical representation of variables which would not have it — seems to corroborate their intuition, although reasoning on types beyond deciding subtyping is currently left as future work.

We now show how this extension of the type system is encoded in our logic.

6.5.3 Logical encoding of variables

We extend the logic with atomic propositions α which behave similarly as σ except they are not mutually exclusive. The interpretation of these propositions is defined as:

$$[\![\alpha]\!] = \{(\sigma_L[tl], c) \mid \iota(\alpha) \in L\}$$

$$[\![\neg\alpha]\!] = \{(\sigma_L[tl], c) \mid \iota(\alpha) \notin L\}$$

The translation $\mathsf{form}(\tau)$ of types into formulas is extended in the obvious way by $\mathsf{form}(\alpha) = \alpha$.

Theorem 6.5.3. *With these extended definitions,* $\mathbb{F}[\![\mathsf{fullform}(\tau)]\!] = [\![\tau]\!]$.

Proof outline: Preliminary remark: whenever φ does not contain any $\langle 2 \rangle$ at toplevel (which is the case of the formulas representing types), then $[\![\varphi]\!] = \mathbb{F}[\![\varphi]\!] \times \mathbf{C}$ where $\mathbf{C}$ is the set of all possible contexts. Hence, when considering such formulas, set-theoretic relations between full interpretations are equivalent to the same relations between first components.

First we check that $\mathbb{F}[\![\mathsf{isd}]\!] = \mathcal{D}$ and reformulate the statement as $\mathcal{D} \cap \mathbb{F}[\![\mathsf{form}(\tau)]\!] = [\![\tau]\!]$.

We make the embedding function tree explicit for greater clarity. What we have to show is that, for any d in $\mathcal{D}$, we have $(d : \tau)$ if and only if $(\mathsf{tree}(d), c)$ is in $[\![\mathsf{form}(\tau)]\!]$ for some (or, equivalently, for any) c.

The property is proved by induction on the pair (d, τ), following the definition of the predicate:

- for $(c : b)$ it holds by definition.
- for $((d_1, d_2)_L : \tau_1 \times \tau_2)$, let $f = (\mathsf{tree}((d_1, d_2)_L), c)$. f is in $[\![\mathsf{form}(\tau_1 \times \tau_2)]\!]$ if and only if $f\langle 1 \rangle$ is in $[\![\mathsf{form}(\tau_1)]\!]$ and $f\langle 1 \rangle\langle 2 \rangle$ is in $[\![\mathsf{form}(\tau_2)]\!]$. (We already know that the node name is $(\times)$ by the structure of d.) Just see that the tree rooted at $f\langle 1 \rangle$ is $\mathsf{tree}(d_1)$ and the one at $f\langle 1 \rangle\langle 2 \rangle$ is $\mathsf{tree}(d_2)$.
- for functions, use the finite unfolding property and the fact the set of pairs is finite, then see, similarly as above, that the correct properties are enforced when navigating the tree.
- for union, negation and empty types, use the preliminary remark.
- for $(d : \alpha)$, just see that $d \in \iota(\alpha)$ and $d \in \mathbb{F}[\![\alpha]\!]$ both mean that the root node of d, which is the node at focus in the formula, bears the label $\iota(\alpha)$.
- for $(d : \mu v.\tau)$, use the property that the interpretation of a fixpoint formula and its unfolding are the same (lemma 4.2 of [Genevès et al., 2007b]).

□

Corollary 6.5.4. *$\tau_1 \leqslant \tau_2$ holds if and only if $\mathsf{fullform}(\tau_1 \wedge \neg\tau_2)$, or alternatively $\mathsf{isd} \wedge \mathsf{form}(\tau_1) \wedge \neg\mathsf{form}(\tau_2)$, is unsatisfiable.*

6.5.4 Complexity

Lemma 6.5.5. *Provided two types τ_1 and τ_2, the subtyping relation $\tau_1 \leqslant \tau_2$ can be decided in time $2^{\mathcal{O}(|\tau_1|+|\tau_2|)}$ where $|\tau_i|$ is the size of τ_i.*

Proof outline: The logical translation of types performed by the function $\mathsf{form}(\cdot)$ does not involve duplication of subformulas of variable size, therefore $\mathsf{form}(\tau)$ is of linear size with respect to $|\tau|$. Since isd has constant size, the whole translation $\mathsf{fullform}(\tau)$ is linear in terms of $|\tau|$. For testing satisfiability of the logical formula, we use the satisfiability-checking algorithm presented in [Genevès et al., 2007b] whose time complexity is $2^{\mathcal{O}(n)}$ in terms of the formula size n. □

6.6 Practical Experiments

In this section we report on some interesting lessons learned from practical experiments with the implementation of the system in order to prove relations in the type algebra.

6.6.1 Implementation

The algorithm for deciding the subtyping relation has been fully implemented on top of the satisfiability solver introduced in Section 4.5.5.

In the polymorphic setting, a counter-example, that is, a model satisfying a formula, is in principle, according to the extended semantics, a labelled tree. However, as mentioned in Section 6.5.2, whenever a formula is satisfiable there always exists an infinity of possible labellings which satisfy it. Therefore, rather than proposing just one labelled tree, the solver gives a minimal tree together with labelling constraints representing all labellings which make that particular tree a counter-example. Namely, for each variable α, every node will be labelled with α to indicate that it must be labelled with α for the formula to be satisfied, with $\neg\alpha$ to indicate that it must not be, or with nothing if label α is irrelevant for that particular node. This allows an easier interpretation of the counter-example in terms of assignments: the subtyping relation fails whenever the assignment for each variable α contains all the trees whose root is marked with α and none of those whose root is marked with $\neg\alpha$.

6.6.2 Concrete Syntax for Type Algebra

All the examples in the subsection that follows can be tested in our online prototype. For this purpose, the following table gives the correspondence between the syntax used in the chapter and the syntax that must be used in the implementation. Additionally, the embedding of a base formula of the logic into a base type is provided by curly braces: $\{\varphi\}$ is an abbreviation for $\mathsf{isbase} \wedge \langle 1 \rangle \varphi$.

	Chapter Syntax	Implementation Syntax
Type variables	α, β, γ	`_a, _b, _g`
Type constructors	$\times, \to$	`*, ->`
Recursive types	$\mu v.\tau$	`let $v = t in $v`
Basic types	$\mathbf{0}, \mathbf{1}$	`F, T`
Logical connectives	$\wedge, \vee, \neg, \Rightarrow$	`&, |, ~, =>`
Subtyping	$\neg(\tau_1 \leqslant \tau_2)$	`nsubtype(t1,t2)`

6.6.3 Examples and Discussion

The goal of this subsection is to illustrate through some examples how our logical setting is natural and intuitive for proving subtyping relations. For example, one can prove simple properties such as the one below:

$$(\alpha \to \gamma) \wedge (\beta \to \gamma) \leqslant (\alpha \vee \beta) \to \gamma \tag{6.9}$$

This is formulated as follows:

```
nsubtype((_a -> _g) & (_b -> _g), (_a | _b) -> _g)
```

```
(mu X8.(((
(let_mu
  X5=(((BASE & <1>(mu X4.(((~(<1>T) | <1>X4) & (~(<2>T) | <2>X4))
     & (~(ERROR) & ~(BASE) & ~(FUNCTION) & ~(PAIR)))))
       | (PAIR & <1>(X5 & <2>(X5 & ~(<2>T))))) | (FUNCTION & (~(<1>T) | <1>X6))),
  X6=(((~(<2>T) | <2>X6) & PAIR) & <1>(X5 & <2>((X5 | (ERROR & ~(<1>T))) & ~(<2>T))))
in
  X5) & ((FUNCTION & (~(<1>T) | <1>(mu X1.((~(<2>T) | <2>X1) & <1>(~(_a) | <2>_g)))))
      & (FUNCTION & (~(<1>T) | <1>(mu X2.((~(<2>T) | <2>X2) & <1>(~(_b) | <2>_g)))))))
       & (~(FUNCTION) | (<1>T & (~(<1>T) | <1>(mu X7.((<2>T & (~(<2>T) | <2>X7))
        | (~(<1>T) | <1>((_a | _b) & (~(<2>T) | <2>~(_g))))))))) | (<1>X8 | <2>X8)))
```

Figure 6.1: Logical translation tested for satisfiability.

which is automatically compiled into the logical formula shown on Figure 6.1 and given to the satisfiability solver that returns:

```
Formula is unsatisfiable [16 ms].
```

which means that no satisfying tree was found for the formula, or, in other terms, that the negation of the formula is valid. The satisfiability solver is seen as a theorem prover since its run built a formal proof that property (6.9) holds.

Jerôme Vouillon [Vouillon, 2006] uses simple examples with lists to illustrate polymorphism with recursive types. For instance, consider the type of lists of elements of type α:

$$\tau_{\text{list}} = \mu v.(\alpha \times v) \vee \text{nil}$$

where "nil" is a singleton type. The type of lists of an even number of such elements can be written as:

$$\tau_{\text{even}} = \mu v.(\alpha \times (\alpha \times v)) \vee \text{nil}$$

By giving the following formula to the solver :

```
nsubtype(let $v = (_a * _a * $v) | {nil} in $v,
               let $w= (_a * $w) | {nil} in $w )
```

which is found unsatisfiable, we prove that

$$\tau_{\text{even}} \leqslant \tau_{\text{list}}$$

If we now consider the type of lists of an odd number of elements of type α:

$$\tau_{\text{odd}} = \mu v.(\alpha \times (\alpha \times v)) \vee (\alpha \times \text{nil})$$

we can check additional properties in a similar manner, like:

$$(\tau_{\text{even}} \vee \tau_{\text{odd}} \leqslant \tau_{\text{list}}) \wedge (\tau_{\text{list}} \leqslant \tau_{\text{even}} \vee \tau_{\text{odd}})$$

The following formula corresponds to the example (6.4) of the introduction:

```
bool() = {true|false};
list() = let $l = (_a * $l) | {nil} in $l;
odd()  = let $o = (_a * _a * $o) | (_a * {nil}) in $o;
even() = let $e = (_a * _a * $e) | {nil} in $e;

nsubtype ( (odd() -> {true}) & (even() -> {false}),
           list() -> bool() )
```

This formula is found unsatisfiable by the solver, which proves the validity of the subtyping statement (6.4).

Giuseppe Castagna (see section 2.7 of [Castagna & Xu, 2011]) gives some examples of non-trivial relations that hold in the type algebra. For instance, the reader can check that the types $\mathbf{1} \to \mathbf{0}$ and $\mathbf{0} \to \mathbf{1}$ can be seen as extrema among the function types:

$$\mathbf{1} \to \mathbf{0} \leqslant \alpha \to \beta \quad \text{and} \quad \alpha \to \beta \leqslant \mathbf{0} \to \mathbf{1}$$

Our system also permitted to detect an error in [Castagna & Xu, 2011] and provided some helpful information to the authors of [Castagna & Xu, 2011] in order to find the origin of the error and make corrections. Specifically, in a former version of [Castagna & Xu, 2011], the following relation was considered:

$$(\neg\alpha \to \beta) \leqslant ((\mathbf{1} \to \mathbf{0}) \to \beta) \vee \alpha \tag{6.10}$$

Authors explained how this relation was proved by their algorithm. However, by encoding the relation in our system we found that this relation actually does not hold. Specifically, this is formulated as follows in our system:

```
nsubtype (~_a -> _b, ((T -> F) -> _b) | _a)
```

The satisfiability solver, when fed this formula, returns the following counter-example:

```
FUNCTION ~_a (PAIR(FUNCTION _a (#,  ~_b ERROR), #), #)
```

`FUNCTION` represents $(\to)$ and `PAIR` represents $(\times)$. This is a binary tree representation of the n-ary tree

$$(\to)_{\neg\alpha}[(\times)[(\to)_{\alpha}[\epsilon] :: \Omega :: \epsilon] :: \epsilon]$$

which corresponds to the domain element

$$\{(\{\}_{\alpha}, \Omega)\}_{\neg\alpha}.$$

The inner $(\to)$ node has no children and thus represents the function which always diverges: $\{\}$. More precisely, it represents a copy f of this function that belongs to the interpretation of α. The root $(\to)$ node then represents a

function which is not in $[\![\alpha]\!]$ and which to f associates an error, while diverging on any other input.

Now, why is it a counter-example to (6.10)? As the function diverges but on one input f and that input is in $[\![\alpha]\!]$, it is vacuously true that on all inputs in $[\![\neg\alpha]\!]$ for which it returns a result, this result is in $[\![\beta]\!]$. Thus it does have the type on the left-hand side. However, it does not have type α, nor does it have type $((\mathbf{1} \to \mathbf{0}) \to \beta)$. Indeed, f does have type $\mathbf{1} \to \mathbf{0}$ and our counter-example function associates to it an error, which is not in $[\![\beta]\!]$.

6.7 Related Work

We review below related works while recalling how the introduction of XML progressively renewed the interests in parametric polymorphism.

The seminal work by Hosoya, Vouillon and Pierce on a type system for XML [Hosoya et al., 2005b] applied the theory of regular expression types and finite tree automata in the context of XML. The resulting language XDuce [Hosoya & Pierce, 2003] is a strongly typed language featuring recursive, product, intersection, union, and complement types. The subtyping relation is decided through a reduction to containment of finite tree automata, which is known to be in EXPTIME. This work does not support function types nor polymorphism, but provided a ground for further research.

In particular, Frisch, Castagna and Benzaken provide a gentle introduction to semantic subtyping in [Frisch et al., 2008]. Semantic subtyping focuses on a set-theoretic interpretation, as opposed to traditional subtyping through direct syntactic rules. Our logical modeling presented in Section 6.4 naturally follows the semantic subtyping approach as the underlying logic has a set-theoretic semantics. Frisch, Castagna and Benzaken added function types to the semantic subtyping performed by XDuce's type system. This notably resulted in the CDuce language [Benzaken et al., 2003]. However, CDuce does not support type variables and thus lacks polymorphism.

Vouillon studied polymorphism in the context of regular types with arrow types in [Vouillon, 2006]. Specifically, he introduced a pattern algebra and a subtyping relation defined by a set of syntactic inference rules. A semantic interpretation of subtyping is given by ground substitution of variables in patterns. The type algebra has the union connective but lacks negation and intersection. The resulting type system is thus less general than ours.

Polymorphism was also the focus of the later work found in [Hosoya et al., 2009]. In [Castagna & Xu, 2011], it is explained that at that time a semantically defined polymorphic subtyping looked out of reach, even in the restrictive setting of [Hosoya & Pierce, 2003], which did not account for higher-order functions. This is why [Hosoya et al., 2009] fell back on a somewhat syntactic approach linked to pattern-matching that seemed difficult to extend to higher-order functions. Our work shows however that such an extension was possible using similar basic ideas, only slightly more abstract.

The most closely related work is the one found in [Castagna & Xu, 2011], in the same proceedings as the current chapter, which solves the problem of

defining subtyping semantically in the polymorphic case for the first time, and addresses the problem of its decision through an ad-hoc and multi-step algorithm, which was only recently proved to terminate in all cases. Our approach also addresses the problem of deciding their subtyping relation and solves it through a more direct, generic, natural and extensible approach since our solution relies on a modeling into a well-known modal logic (the μ-calculus) and on using a satisfiability solver such as the one proposed in [Genevès et al., 2007b]. This logical connection also opens the way for extending polymorphic types with several features found in modal logics.

The work of [Bierman et al., 2010] follows the same spirit than ours: type-checking is subcontracted to an external logical solver. An SMT-solver is used to extend a type-checker for the language Dminor (a core dialect for M) with refinement type and type-tests. The type-checking relies on a semantic subtyping interpretation but neither function types nor polymorphism are considered. Therefore, their work is incomparable to ours.

The present work heavily relies on the work presented in [Genevès et al., 2007b] since we repurpose the satisfiability-checking algorithm of [Genevès et al., 2007b] for deciding the subtyping relation. The goal pursued in [Genevès et al., 2007b] was very different in spirit: the goal was to decide containment of XPath queries in the presence of regular tree types. To this end, the decidability of a logic with converse for finite ordered trees is proved in a time complexity which is a simple exponential of the size of the formula. The present work builds on these results for solving semantic subtyping in the polymorphic case.

6.8 Conclusion

The main contribution of this chapter is to define a logical encoding of the subtyping relation defined in [Castagna & Xu, 2011], yielding a decision algorithm for it. We prove that this relation is decidable with an upper-bound time complexity of $2^{(n)}$, where n is the size of types being checked. In addition, we provide an effective implementation of the decision procedure that works well in practice.

This work illustrates a tight integration between a functional language type-checker and a logical solver. The type-checker uses the logical solver for deciding subtyping, which in turn provides counter-examples (whenever subtyping does not hold) to the type-checker. These counterexamples are valuable for programmers as they represent evidence that the relation does not hold. As a result, our solver represents a very attractive back-end for functional programming languages type-checkers.

This result pushes the integration between programming languages and logical solvers to an advanced level. The proposed logical approach is not only capable of modeling higher order functions, but it is also capable of expressing values from semantic domains that correspond to monadic second-order logics such as XML tree types. This shows that such logical solvers can become the

core of XML-centric functional languages type-checkers such as those used in CDuce or XDuce.

Seven

Conclusion & perspectives

7.1 Contributions summary

The guiding motivation of my work is to promote declarative and typed representation of content in documents and web applications. The goal is to ease the design of web applications and make them richer, safer and more efficient. During the last few years, I focused first on enriching content representation to support a wider and more integrated set of features: temporal synchronization, spatial positioning, logical organization and hypermedia links. The central idea of my work consists in defining the document through various dimensions or facets with well-defined languages. In particular, one of my first contributions was the introduction of such languages for the temporal dimension of documents and making it seamlessly integrated with the other dimensions. Then, I have introduced supportive methods for time-based languages such as intelligent scheduling and enhanced runtime support. My work on temporal synchronization resulted in a W3C recommended language called SMIL. This language has been used in mobile infrastructure for Multimedia Messaging and integrated in other languages such as SVG.

Then I have considered the problem of content adaptation raised by the double evolution of the web: by the introduction of various types of both terminals and communications media channels. Mobile phone networks such as third generation networks and WiFi have pushed the boundaries of the web infrastructure much further from its initial vision. At the same time, more and more devices became connected such as cellular phones, tablets, television sets, game consoles and all sorts of embedded devices. In response to this evolution, I have proposed web access frameworks based on negotiation and automatic adaptation methods. More precisely, I have explored two adaptation approaches; one based on structural transformations operating on a more advanced web architecture, the other on presentation "semantics". My overall goal was to enforce a vision of the web were content is accessible for all anywhere anytime and to avoid web fragmentation.

Then my focus shifted to content design and production. In particular, I proposed an incremental framework aimed at enhancing our ability to produce content easily but also to design transformations capable of generating content presentation automatically. Transformations are crucial as they are today frequently used on content servers to produce content presentations at access time.

I have shown that efficient incremental transformations rely on precise static analyses of types and path expressions in these transformations. These analyses are not only limited to performance issues but are also critical for ensuring one of the key transformation properties: safety. More precisely, type safety consists in guaranteeing using some decision procedures that transformations always produce valid documents against their schema.

The initial idea of this work consists in considering that two issues need to be answered in order to solve such decision problems in XML. First, it was essential to identify an appropriate formal framework with sufficient expressive power to capture both regular tree languages and navigation introduced by XPath. Then we had to find appropriate techniques to solve the satisfiability problem in that framework. Satisfiability allows determining whether a given property holds or not. Then, it would be interesting to obtain an XML document that exemplifies it. Such properties include XPath containment, emptiness, equivalence and coverage of XPath queries (in the presence or absence of regular types of trees).

A first important result was achieved through the design of a finite tree logic adapted to XML and its decision procedure. The logic is expressive enough to capture regular tree types along with multi-directional navigation in finite trees. It is decidable in single exponential time (specifically in $2^{O(n)}$ steps where n is the size of the input formula defined as its number of atomic propositions and eventualities). This improves the best-known computational complexity for finite trees. Another contribution is that we showed how to linearly compile queries and regular tree types (including DTDs and XML Schemas) in the logic. This offers a uniform notation for both constructs and facilitates reasoning on them. The logic enjoys the nice property of being closed under boolean operations. It supports the full navigational features of XPath and covers the largest fragment considered in the literature [Marx, 2004]. From the algorithmic point of view, the decision procedure proved entirely feasible using symbolic techniques borrowed from verification: BDD (Binary Decision Diagrams).

In a follow-up work, I have considered with colleagues the problem of XML Schema evolution. In the ever-changing context of the web, XML schemas continuously change in order to cope with the natural evolution of entities they describe. Schema changes have important consequences. First, existing documents valid with respect to the original schema are no longer guaranteed to fulfill the constraints described by the evolved schema. Second, the evolution also impacts programs manipulating documents whose structure is described by the original schema.

I explored unifying frameworks for determining the effects of XML Schema evolution both on the validity of documents and on queries. The result is a powerful system capable of analyzing various scenarios in which forward/backward compatibility of schemas is broken, and in which the result of a query may not be anymore what was expected. Specifically, the system offers a predicate language, which allows one to formulate properties related to schema evolution. The system then relies on exact reasoning techniques to perform a fine-grained

analysis of programs and schema changes. This yields either a formal proof of the property or a counter-example that can be used for debugging purposes. The system has been fully implemented and tested with real-world use cases, in particular with the main standard document formats used on the web, as defined by W3C. The system identifies precisely compatibility relations between document formats. In case these relations do not hold, the system can identify queries that must be reformulated in order to produce the expected results across successive schema versions. The long-term goal of this work is to find methods and techniques to reformulate XML transformations automatically when schemas evolve.

In a more recent work, I have investigated the means to obtaining powerful type systems where types can denote not only data types such as schemas but also computations. To that end, I studied how function types can be supported in our logic in the manner of [Benzaken et al., 2003; Castagna & Xu, 2011]. In addition, if functions can be made parametric using variables (parametric types), they become more generic since they can operate on a large number of specific types. Such functions are also important to promote code reuse. To that end, parametric polymorphism is studied for type systems aiming at maintaining full static type-safety of functional programs that manipulate linked structures such as trees, potentially with higher-order functions. To that end, advanced type algebra equipped with recursive, product, function (arrow), intersection, union, and complement types is introduced. I first show how the subtyping relation between such type expressions can be decided through a purely logical approach.

The main result along this research direction solves an open problem: we prove the decidability of the subtyping relation when this type algebra is extended with type variables. This provides a powerful polymorphic type system (using ML-style prenex polymorphism, where variables are implicitly universally quantified at top level), for which defining the subtyping relation is not obvious, as pointed out in [Castagna & Xu, 2011], and for which no candidate definition of subtyping had been proved decidable before. The novelty, originality and strength of our solution reside in introducing a logical modeling for the semantic subtyping framework. Specifically, semantic subtyping is modeled in the finite tree logic presented earlier and rely on a slightly modified satisfiability solver in order to decide subtyping in practice. An EXPTIME ($2^{O(n)}$) complexity bound is obtained as well as an efficient implementation in practice.

There are a number of directions for future work, from the results presented above one can foresee more ambitious and unified end-to-end programming models covering content description, processing and distribution. At a higher level of abstraction, applications can be seen as a set of distributed functions connected by web services. They can be analyzed for non-functional aspects such as performance by code and data distribution, security and privacy enforcement while enhancing their scalability on increasingly popular infrastructures such as the cloud.

7.2 Perspectives

The perspectives of my research work are drawn here in the form of a research project I am currently proposing, as a project leader, with colleagues at LIG Laboratory. Pierre Genevès, Cécile Roisin, Nils Gesbert and Jacques Lemordant are taking part of this new adventure.

7.3 Motivations: social and economic challenges

During the last two decades, the web became crucial in our daily life activities (work, banking, shopping, education, administration, leisure, social networking). The web is now by far the largest mass of information that mankind has ever gathered. This revolution is continuing its path toward a more compelling user experience through richer content (such in HTML5, multimedia, 3D audio and graphics) and ever increasing web applications via their reconversion through services. The future of the web will be influenced by our ability to leverage this unprecedented potential and to accomplish the successful synergy of applications and richer content.

This fundamental revolution has been witnessed by a shift in the web itself that moved from a rather static hypertext system, to a more dynamic environment where content combined in web applications became the standard, not only in content dissemination but also in application development. A new generation of applications such as enterprise application software, collaborative online publishing platforms, social networking, rating, shopping, mobile navigation via augmented reality are reshaping the IT industry landscape.

This rapid revolution has been made possible mainly by hacking old technology both for content and software that has now showed its severe limitations. Web content is becoming difficult to integrate and extend to richer features and applications are becoming harder to write, maintain, and evolve. As a consequence, information built on such infrastructure has never been at such a level of risk of a digital black hole. These facts are severely undermining a web of trust, where technologies remain open, public and applications are made reliable, secure and efficient.

This proposal aims at developing a vision of a web where content is enhanced and protected, applications made easier to build, maintain and secure. We seek at opening new horizons for the development of the web, enhancing its potential, effectiveness, and dependability. In particular, we aim at making significant contributions by obtaining fundamental results, building advanced experimental applications showcasing these results and by contributing to Web standards. The challenging part here is that contributing to each of these lines of work requires progress in all of them simultaneously.

7.4 Objectives in terms of technology

Despite the major social and economic challenges that the evolution of the web represents, current content representation practices and programming methods are severely limited. Designing web applications is becoming increasingly

complex as it relies more and more on a jungle of programming languages, tools and data formats, each targeted toward the different application layers (presentation, application and storage). This often yields complex and opaque applications organized in silos, which are costly, inefficient, hard to maintain and evolve, and vulnerable to errors and security holes. In addition, the communication aspects are often handled independently via remote service invocations and hidden from the verification aspect. As a consequence, there is an urgent need and a growing demand for a uniform programming framework that captures the essence of web applications: advanced content, data and communication. Furthermore, successful candidate frameworks must capture rich document formats, data models and communication patterns. A crucial aspect is to offer correction guarantees and flexibility in the application architecture. For instance, applications need to be checked, optimized and managed as a whole while leveraging on the consistency of their individual components and data fragments.

7.5 Scientific goals and research directions

The main open problem that we can observe today is a lack of formalisms, concepts and tools for reasoning simultaneously over documents and communication aspects in programs. The scientific challenge that we face is to establish such a unifying framework in the context of the web. This is a difficult problem that we propose to address along three complementary directions:

a) **design of advanced web applications**, which consists in building a new generation of multimedia and augmented reality applications with new design foundations. The challenging part is to propose means to combine in an easy and compositional manner rich content, augmented reality and data gathered dynamically such as those stemming from environment and from sensors.

b) **modeling**, which consists in capturing various aspects of document processing, data and communication in a unifying model, and whose difficult part consists of taking into account the peculiarities of the web that require new programming models and the supporting theoretical tools that do not exist today.

c) **analysis, verification and optimization**, which consist in guaranteeing safety and efficiency properties of information systems, and whose hard part consists in dealing with problems close to the frontier of decidability, and therefore in finding useful balances between programming ease, expressivity, complexity, succinctness, algorithmic techniques and effective implementations.

This research proposal aims at developing content models, formalisms, languages, concepts, algorithms, and tools for building a unifying framework, along the three directions above. These directions are closely related and interdependent. We intend to make contributions to each of them first, by obtaining

fundamental results, second, by building advanced experimental applications and tools and third, by contributing to Web standards. The overall goal is to enable richer, more reliable, secure, and efficient systems. We give more details on each direction below.

7.5.1 Design of advanced web applications

Specificity of web documents: ordered tree structures

Web documents (and in particular XML) provide a new field of study, for which it is not possible to use already existing techniques without substantial modifications. The peculiarity of web documents originates from their ordered tree structure. These structures can for example be seen as a relaxation of the classical relational model, one of the foundations of traditional databases, where less rigid and homogeneous "data fields" are allowed. This data model has proven to be very useful for representing various families of documents: multimedia, hypertext, news articles, scientific documents, etc. However, it needs to be revisited to account for richer and more dynamic content. It is therefore necessary to develop new theoretical foundations, possibly drawing on methods used in other domains of computer science.

One essential concept consists in describing classes of documents that share the same requirements (e.g. web pages through XHTML, or mathematical formulas through MathML). Mastering such representations and their interactions is also crucial for reasoning over sets of documents. From a theoretical point of view, this modeling task constitutes a renewal for the study of tree automata and logical theories introduced in the late 1960's. These theories are rapidly evolving to support the new features provided by web documents, requiring more and more expressiveness and succinctness. We intend to contribute to this modeling effort, especially through contributions on modal logics such as the modal μ-calculus, introduced more recently.

Universal content models and formats

Models and formats used for sharing multimedia content on the web must represent the many facets of multimedia documents. Their richness and versatility determine how multimedia content can be processed and used in various contexts. During the last decade, content has shifted from mainly static pages to highly dynamic and programmable ones. However, content was massively produced in a hackish manner and very basic document features have been subcontracted to scripting which became the "Jack of all Trades" technology in browsers. In addition, a huge portion of the content on the web is today not represented in an adequate manner, severely compromising their long-term access and automatic processing.

It is vital to design rigorously documents to outlive any particular piece of hardware or system where they may reside. Furthermore, we seek to build advanced document models that allow us to describe the increasing variety of modalities such as 3D sound, augmented reality and dynamic content (e.g. data streams), which are becoming a commodity on mobile platforms and ap-

plications. The difficulty here is to be able to create models and formats that combine these aspects by declarative means in a consistent manner, both at syntactic and semantic levels. They should be able to both enhance user experience and facilitate their manipulation by programs.

Supporting integrated, rich, dynamic and augmented content

Until now, content rendering on the web was mainly based on supporting media formats separately. It is still notably the case in HTML5 where vector graphics, mathematical content, audio and video are supported as isolated media types. With their increasing support in browsers together with others such as 3D audio and graphics, we need more than ever methods to integrate them tightly and correctly in applications and in particular in browsers. In addition, with the increasing use of web content in mobile terminals, we need to take into account highly dynamic information flowing from sensors (positioning and orientation move) and camera. This information needs to be captured and efficiently combined with content in web browsers. To reach that goal, we need to ease the manipulation of such content with carefully designed programming interfaces and by developing supporting integrative methods. The challenge is to find appropriate abstractions while hiding the increasing complexity of such content.

7.5.2 Modeling documents, data and communications

The web is traditionally composed of resources (data and documents) and services (applications) that exchange resources. The frontier between the two becomes fuzzy as more and more scripting occurs in web pages. However, scripting is currently done at a very low level (e.g. similar to an assembly language) and this prevents many sorts of analysis and processing. If we consider XML programming at a higher level then, in the same way as XML documents are twofold — the raw content and its type — we can consider two aspects of a programming language, with respect to XML: whether or not it provides syntactic support to process XML documents (content side), whether or not it can enforce document constraints (type side) and, finally whether or not it offers the means to integrate smoothly with external services (communication side). We believe that there is a need for higher-level abstractions that make machine processing possible or easier, and that integrate/encompass all these aspects. Current programming technology is still very limited from this perspective. For example, XQuery, which is a good candidate for a uniform and high-level language, has a very imprecise type system and has no communication facilities. It is also agnostic to some important content facets such as style, layout, synchronization and dynamics.

More generally, representing in a uniform way data and programs is a first step toward higher order programming, as noticed by Luca Cardelli in his work on semi-structured computation when he remarks: “if we can take advantage of the similarities [between mobile computation and semi-structured data] and generalize them, we may obtain a broader model of data and computation on

the Internet." [1] We believe that this important step can be investigated along several directions. One direction consists of extending expressive modal logics with, for instance, function types for representing programs. Another direction consists of considering process calculi as the missing part for, e.g., extending the XQuery data model with a broader computation model accounting for higher order capabilities such as parametric polymorphism, functions, etc.

7.5.3 Analysis by reasoning

Type-checking web applications

Developing safer web applications depends on the quality of the methods, that we will be able to produce in order to enable the correct manipulation of data in applications. Classical program verification techniques fail to extend to rich data manipulations, which are the core of web programming.

We propose to develop web program analysis techniques (verification and optimization), which allow the detection of errors and the enhancement of performance in data manipulation. We will concentrate on techniques based on type checking by introducing appropriate type systems and reasoning techniques on programs. The main challenge here is to find decidable methods whose complexity does not preclude their practical applicability. To reach this goal, we intend to use logical methods such as modal logics and satisfiability solvers where we gained significant experience.

Global verification of data manipulation and exchange

We seek to build global analysis and verification techniques encompassing errors in data manipulation, data exchanges in communication protocols and the interactions between application components. The type systems approach seems particularly appropriate since it allows, by construction, to ensure global properties of a web application by a local and modular verification of its components. As such, it will constitute an important object of investigation. Specifically, we will focus on type systems based on the modeling described in Section 7.5.2. The expected benefit of such a formalization is to leverage on the extension of the large toolbox of proof methods and theoretical results that equip existing calculi. Ideas, concepts, and techniques from process calculi have been successfully applied to the study of behavioral properties of distributed systems, and of type systems for concurrent (functional and/or object-oriented) languages. Overall, we seek to integrate all these aspects in a uniform and sound type system.

Designing for evolution

In the ever-changing context of the web, XML schemas continuously change in order to cope with the natural evolution of the entities they describe. A change in the schema may require updates of programs that must cope with

[1] Luca Cardelli. "Semistructured Computation". In Research Issues in Structured and Semistructured Database Programming. Lecture Notes in Computer Science, Volume 1949, 2000.

the newly described set of valid documents. We propose to introduce new methods and tools for determining and facilitating program updates resulting from these changes. Similarly, web services evolve over time, through the modifications of their interfaces. We intend to develop reasoning techniques capable of analyzing programs, schemas and communications in order to automate and efficiently guide these unavoidable updates. Such methods are crucial to enforce quality assurance in web applications and to help tackling forward/backward compatibilities issues.

Bibliography

3GPP (2009). Multimedia messaging service (MMS), 3GPP specification, 3GPP TS 23.140. http://www.3gpp.org/ftp/Specs/html-info/23140.htm. 14

3GPP (2011). Transparent end-to-end packet-switched streaming service (pss); 3gpp smil language profile, 3GPP specification, 3GPP TS 26.246. http://www.3gpp.org/ftp/Specs/html-info/26246.htm. 14

Abdeddaim, Y., Asarin, E., & Sighireanu, M. (2009). Simple algorithm for simple timed games. In Proceedings of the 2009 16th International Symposium on Temporal Representation and Reasoning, TIME '09, (pp. 99–106)., Washington, DC, USA. IEEE Computer Society. 15

Afanasiev, L., Blackburn, P., Dimitriou, I., Gaiffe, B., Goris, E., Marx, M., & de Rijke, M. (2005). PDL for ordered trees. Journal of Applied Non-Classical Logics, 15(2), 115–135. 44

Akpotsui, E. (1993). Transformation de types dans les sytèmes d'édition de documents structurés. Thèse de doctorat, Institut National Polytechnique de Grenoble. 21

Akpotsui, E. & Quint, V. (1992). Type transformation in structured editing systems. In EP92 Proceedings of International Conference on Electronic Publishing, Document Manipulation, and Typography, (pp. 27–41). EPFL, Switzerland. 21

Akscyn, R. M., McCracken, D. L., & Yoder, E. A. (1988). Kms: a distributed hypermedia system for managing knowledge in organizations. Commun. ACM, 31, 820–835. 2

Altisen, K., Gößler, G., Pnueli, A., Sifakis, J., Tripakis, S., & Yovine, S. (1999). A framework for scheduler synthesis. In IEEE Real-Time Systems Symposium, (pp. 154–163). 15

André, J., Furuta, R., & Quint, V. (1989). By way of an introduction. Structured documents: what and why?, (pp. 1–6). New York, NY, USA: Cambridge University Press. 9

ATA (1997). Ata specification 2100, digital standards for aircraft support, revision 3. air transport association of america. Standard. 10, 20

Ayars, J., Bulterman, D., Cohen, A., Day, K., Hodge, E., Hoschka, P., Hyche, E., Jourdan, M., Kim, M., Kubota, K., Lanphier, R., Layaïda, N., & al (2001). Synchronized multimedia integration language (SMIL 2.0). W3C recommendation, World Wide Web Consortium. 13

Bamford, R., Borkar, V. R., Brantner, M., Fischer, P. M., Florescu, D., Graf, D. A., Kossmann, D., Kraska, T., Muresan, D., Nasoi, S., & Zacharioudaki, M. (2009). Xquery reloaded. PVLDB, 2(2), 1342–1353. 25

Bárcenas, E., Genevès, P., Layaïda, N., & Schmitt, A. (2011). Query reasoning on trees with types, interleaving, and counting. In IJCAI 2011: Proceedings of the 22nd International Joint Conference on Artificial Intelligence, (pp. 718–723). IJCAI/AAAI. 21

Bárcenas-Patiño, E. (2011). Automated reasoning on trees with cardinality constraints. Phd thesis, Institut National Polytechnique de Grenoble. 82, 89

Benedikt, M. & Cheney, J. (2010). Destabilizers and independence of XML updates. Proceedings of the VLDB Endowment, 3(1), 906–917. 125

Benedikt, M., Fan, W., & Geerts, F. (2005). XPath satisfiability in the presence of DTDs. In PODS '05: Proceedings of the twenty-fourth ACM Symposium on Principles of Database Systems, (pp. 25–36)., Baltimore, Maryland. ACM Press. 78, 121

Benedikt, M. & Koch, C. (2009). XPath leashed. ACM Comput. Surv., 41, 3:1–3:54. 121

Benzaken, V., Castagna, G., & Frisch, A. (2003). CDuce: An XML-centric general-purpose language. In ICFP '03: Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming, (pp. 51–63)., Uppsala, Sweden. ACM Press. 22, 24, 82, 122, 125, 126, 143, 149

Berglund, A. (2006). Extensible stylesheet language (XSL) version 1.1, W3C recommendation. http://www.w3.org/TR/xsl/. 9, 10, 11

Berners-Lee, T., Cailliau, R., Groff, J.-F., & Pollermann, B. (1992). World-wide web: The information universe. Electronic Networking: Research, Applications and Policy, 1(2), 74–82. 2

Beyer, K., Özcan, F., Saiprasad, S., & der Linden, B. V. (2005). DB2/XML: designing for evolution. In SIGMOD '05, (pp. 948–952). ACM. 121

Bierman, G. M., Gordon, A. D., Hriţcu, C., & Langworthy, D. (2010). Semantic subtyping with an SMT solver. In Proceedings of the 15th international conference on functional programming (ICFP '10), (pp. 105–116)., Baltimore, MD, USA. 125, 144

Boag, S., Chamberlin, D., Fernàndez, M. F., Florescu, D., Robie, J., & Siméon, J. (2006). XQuery 1.0: An XML query language, W3C candidate recommendation. http://www.w3.org/TR/xquery/. 2, 3, 23, 78, 82, 124

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., & Yergeau, F. (2004). Extensible markup language (XML) 1.0 (third edition), W3C recommendation. http://www.w3.org/TR/2004/REC-xml-20040204/. 2, 9, 10

Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers, 35(8), 677–691. 66

Bulterman, D., Grassel, G., Jansen, J., Koivisto, A., Layaïda, N., Michel, T., Mullender, S., & Zucker, D. (2005). Synchronized multimedia integration language (SMIL 2.1). W3C recommendation, World Wide Web Consortium. 13

Bulterman, D. C. (2001). Smil 2.0: Overview, concepts, and structure. IEEE Multimedia, 8, 82–88. 14

Calvanese, D., De Giacomo, G., Lenzerini, M., & Vardi, M. Y. (2008). Regular xpath: Constraints, query containment and view-based answering for xml documents. In Proc. of the 2008 Int. Workshop on Logic in Databases (LID 2008). 45

Calvanese, D., De Giacomo, G., Lenzerini, M., & Vardi, M. Y. (2009). An automata-theoretic approach to regular xpath. In Proc. of the 12th Int. Symposium on Database Programming Languages (DBPL 2009), volume 5708 of Lecture Notes in Computer Science, (pp. 18–35). Springer. 45

Calvanese, D., Giacomo, G. D., Lenzerini, M., & Vardi, M. Y. (2010). Node selection query languages for trees. In Fox, M. & Poole, D. (Eds.), AAAI. AAAI Press. 45

Castagna, G. & Nguyen, K. (2008). Typed iterators for XML. In ICFP, (pp. 15–26). 122

Castagna, G. & Xu, Z. (2011). Set-theoretic foundation of parametric polymorphism and subtyping. In Proceedings of the 16th international conference on functional programming (ICFP '11), Tokyo. 6, 24, 123, 126, 133, 134, 135, 136, 138, 142, 143, 144, 149

Cazenave, F., Quint, V., & Roisin, C. (2011). Timesheets.js: when smil meets html5 and css3. In Proceedings of the 11th ACM symposium on Document engineering, DocEng '11, (pp. 43–52)., New York, NY, USA. ACM. 14

Christensen, E., Curbera, F., Meredith, G., & Weerawarana, S. (2001). Web services description language (WSDL) 1.1, W3C recommendation. http://www.w3.org/TR/wsdl. 11

Clark, J. & DeRose, S. (1999). XML path language (XPath) version 1.0, W3C recommendation. http://www.w3.org/TR/ 1999/REC-xpath-19991116. 82, 83, 98, 125

Clark, J. & Murata, M. (2001). RELAX NG specification, OASIS committee specification. http://relaxng.org/spec-20011203.html. 10

Clarke, E. M. & Emerson, E. A. (1981). Design and synthesis of synchronization skeletons using branching-time temporal logic. In Logic of Programs, Workshop, volume 131 of LNCS, (pp. 52–71). Springer-Verlag. 43

Clement, L., Hately, A., von Riegen, C., & Rogers, T. (2004). UDDI Version 3.0.2, UDDI spec technical committee draft, oasis. http://uddi.org/pubs/uddi_v3.htm. 11

Colazzo, D., Ghelli, G., Manghi, P., & Sartiani, C. (2004). Types for path correctness of XML queries. In ICFP '04: Proceedings of the ninth ACM SIGPLAN international conference on Functional programming, (pp. 126–137)., Snow Bird, UT, USA. ACM Press. 121

Colazzo, D., Ghelli, G., Manghi, P., & Sartiani, C. (2006). Static analysis for path correctness of XML queries. Journal of Functional Programming. To appear. 121

Conklin, J. (1987). Hypertext: An introduction and survey. Computer, 20, 17–41. 2

Dahlström, E., Dengler, P., Grasso, A., Lilley, C., McCormack, C., Schepers, D., & Watt, J. (2011). Scalable vector graphics (SVG) 1.1 (second edition), W3C recommendation. http://www.w3.org/TR/SVG11/. 14

de Moura, L. M. & Bjørner, N. (2008). Z3: An efficient SMT solver. In Proceedings of the 14th international conference on tools and algorithms for the construction and analysis of systems (TACAS '08), (pp. 337–340)., Budapest. 125

Draper, D., Dyck, M., Fankhauser, P., Fernández, M., Malhotra, A., Rose, K. H., Rys, M., Siméon, J., & Wadler, P. (2010). XQuery 1.0 and XPath 2.0 Formal Semantics (second edition). W3C Recommendation. http://www.w3.org/TR/2005/WD-xquery-semantics-20050915/. 25

Edmund M. Clarke, J., Grumberg, O., & Peled, D. A. (1999). Model checking. MIT Press. 66, 68, 69

Engovatov, D. & Robie, J. (2010). XQuery 3.0 requirements, W3C working draft. 4, 23, 124

Etemad, E. J. (2010). Cascading style sheets (CSS) snapshot 2010, W3C working group note. http://www.w3.org/TR/css-2010/. 14

Euzenat, J., Layaïda, N., & Dias, V. (2003). A semantic framework for multimedia document adaptation. In IJCAI 2003: Proceedings of the 18th International Joint Conference on Artificial Intelligence, (pp. 31–36). Morgan Kauffman. 18

Fargier, H., Jourdan, M., Layaïda, N., & Vidal, T. (1998). Using temporal constraints networks to manage temporal scenario of multimedia documents. In Proceedings of the ECAI-98 International Workshop on Spatial and Temporal Reasoning, (pp. 1–6). 14, 15

Fielding, R. T. (2000). REST: Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine. 7

Fischer, M. J. & Ladner, R. E. (1979). Propositional dynamic logic of regular programs. Journal of Computer and System Sciences, 18(2), 194–211. 43

Frisch, A., Castagna, G., & Benzaken, V. (2008). Semantic subtyping: dealing set-theoretically with function, union, intersection, and negation types. Journal of the ACM, 55(4), 1–64. 124, 125, 126, 128, 133, 143

Gapeyev, V., Garillot, F., & Pierce, B. C. (2006). Statically typed document transformation: An Xtatic experience. In PLAN-X 2006: Proceedings of the International Workshop on Programming Language Technologies for XML, volume NS-05-6 of BRICS Notes Series, (pp. 2–13)., Charleston, South Carolina, United States. BRICS. 122

Genevès, P. (2006). Logics for XML. PhD thesis, Institut National Polytechnique de Grenoble. http://www.pierresoft.com/pierre.geneves/phd.htm. 104

Genevès, P. & Layaïda, N. (2006a). A satisfiability solver for XML and XPath decision problems. http://wam.inrialpes.fr/xml/. 66, 74, 96, 100, 120

Genevès, P. & Layaïda, N. (2006b). A system for the static analysis of XPath. ACM Transactions on Information Systems, 24(4), 475–502. 21

Genevès, P. & Layaïda, N. (2006c). A system for the static analysis of XPath. ACM Transactions on Information Systems (TOIS), 24(4). 73

Genevès, P. & Layaïda, N. (2007). Deciding XPath containment with MSO. Data and Knowledge Engineering, 63(1), 108–136. 21

Genevès, P. & Layaïda, N. (2011). Inconsistent path detection for XML IDEs. In ICSE 2011: Proceedings of the 33rd International Conference on Software Engineering, (pp. 983–985). ACM. 22

Genevès, P., Layaïda, N., & Quint, V. (2009). Identifying query incompatibilities with evolving XML schemas. In ICFP 2009: Proceeding of the 14th ACM SIGPLAN International Conference on Functional Programming, (pp. 221–230). ACM. 22

Genevès, P., Layaïda, N., & Quint, V. (2009). Identifying query incompatibilities with evolving XML schemas. In ICFP '09: Proceedings of the ACM SIGPLAN international conference on Functional programming, (pp. 221–230). 95

Genevès, P., Layaïda, N., & Quint, V. (2011). Impact of XML schema evolution. ACM Transactions on Internet Technologies, 11(1), 4–27. 22

Genevès, P., Layaïda, N., & Quint, V. (2012). On the analysis of cascading style sheets (to appear). In WWW 2012: Proceedings of the 21st International World Wide Web Conference. ACM. 21

Genevès, P., Layaïda, N., & Schmitt, A. (2007a). Efficient static analysis of XML paths and types. In PLDI 2007: Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation, (pp. 342–351). ACM. 21

Genevès, P., Layaïda, N., & Schmitt, A. (2007b). Efficient static analysis of XML paths and types. In PLDI '07, (pp. 342–351). ACM Press. 45, 46, 80, 101, 102, 103, 104, 112, 121, 122, 125, 126, 129, 130, 133, 139, 144, 162

Genevès, P., Layaïda, N., & Schmitt, A. (2008). Efficient static analysis of XML paths and types. Long version of [Genevès et al., 2007b], Research Report 6590, INRIA. 96, 98, 105

Gesbert, N., Genevès, P., & Layaïda, N. (2011). Parametric polymorphism and semantic subtyping: the logical connection. In ICFP 2011: Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming, (pp. 107–116). ACM. 23

Goldfarb, C. (1996). The roots of sgml, a personal recollection. 9

Gößler, G. (1998). Modèlisation et contrôle des systèmes multimédia. Mémoire de dea, Université Joseph-Fourier - INPG, Laboratoire VERIMAG. 15

Grädel, E., Thomas, W., & Wilke, T. (Eds.). (2002). Automata logics, and infinite games: a guide to current research. Springer-Verlag. 44

Groppe, J. & Groppe, S. (2008). Filtering unsatisfiable XPath queries. Data Knowl. Eng., 64(1), 134–169. 121

Groppe, S., Bottcher, S., & Groppe, J. (2006). XPath query simplification with regard to the elimination of intersect and except operators. In ICDEW '06: Proceedings of the 22nd International Conference on Data Engineering Workshops, (pp.8̃6)., Washington, DC, USA. IEEE Computer Society. 121

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., Nielsen, H. F., Karmarkar, A., & Lafon, Y. (2007). SOAP version 1.2 part 1: Messaging framework (second edition), W3C recommendation. http://docs.oasis-open.org/wsbpel/. 11

Hagimont, D. & Layaïda, N. (2002). Adaptation de serveur multimédia par un code mobile. Technique et Science Informatiques (TSI), numéro spécial Agents et code mobile, 21(6), 877–898. 14, 16

Halasz, F. & Schwartz, M. (1994). The dexter hypertext reference model. Commun. ACM, 37, 30–39. 2

Hickson, I. (2011). HTML5: A vocabulary and associated APIs for HTML and XHTML, W3C working draft. http://www.w3.org/TR/html5/. 14

Hojati, R., Krishnan, S. C., & Brayton, R. K. (1996). Early quantification and partitioned transition relations. In ICCD '96: Proceedings of the 1996 International Conference on Computer Design, VLSI in Computers and Processors, (pp. 12–19). 69

Holtman, K. & Mutz, A. (1998). HTTP remote variant selection algorithm – RVSA/1.0. Experimental standard, IETF. 18

Hoschka, P. (1998). Synchronized multimedia integration language (SMIL) 1.0 specification, W3C recommendation. http://www.w3.org/TR/REC-smil/. 79

Hoschka, P., Bugaj, S., Bulterman, D., Layaïda, N., & al (1998). Synchronized multimedia integration language (SMIL) 1.0 specification. W3C recommendation, World Wide Web Consortium. 13

Hosoya, H., Frisch, A., & Castagna, G. (2009). Parametric polymorphism for XML. ACM Transactions on Programming Languages and Systems, 32(1), 1–56. 133, 134, 135, 143

Hosoya, H. & Pierce, B. C. (2003). Xduce: A statically typed xml processing language. ACM Trans. Internet Technol., 3, 117–148. 22, 122, 126, 143

Hosoya, H., Vouillon, J., & Pierce, B. C. (2005a). Regular expression types for XML. ACM Trans. Program. Lang. Syst., 27(1), 46–90. 89

Hosoya, H., Vouillon, J., & Pierce, B. C. (2005b). Regular expression types for XML. ACM Transactions on Programming Languages and Systems, 27(1), 46–90. 96, 98, 125, 143

Huet, G. P. (1997). The zipper. Journal of Functional Programming, 7(5), 549–554. 47

ISO (2006a). ISO/IEC 19757-3:2006 information technology – document schema definition language (DSDL) – part 3: Rule-based validation – schematron, ISO standard. http://www.schematron.com/. 10

ISO (2006b). ISO/IEC 19757-4:2006 information technology – document schema definition languages (DSDL) – part 4: Namespace-based validation dispatching language (NVDL), ISO standard. http://www.schematron.com/. 10

ISO-SGML (1986). Information processing – text and office systems – standard generalized markup language (SGML). 9

ISO/IEC (1996). Document style semantics and specification language (DSSSL). International Standard ISO/IEC 10179:1996. 10

ISO/IEC (1997). Information processing – Hypermedia/Time-based Structuring Language (HyTime). International Standard ISO/IEC JTC 1/SC 18 WG8 N1920rev. Version 1 known as ISO/IEC 10744:1992. 9

Jordan, D. & Evdemon, J. (2007). Web services business process execution language version 2.0, oasis standard. http://docs.oasis-open.org/wsbpel/. 11

Jourdan, M., Layaïda, N., Roisin, C., Sabry-Ismaïl, L., & Tardif, L. (1998). Madeus, and authoring environment for interactive multimedia documents. In ACM Multimedia 1998: Proceedings of the Sixth ACM International Conference on Multimedia, (pp. 267–272). ACM. 12

Kappe, F., Maurer, H., & Scherbakov, N. (1993). Hyper-g – a universal hypermedia system. Journal of Educational Multimedia and Hypermedia, 2, 39–66. 2

Kim, M., Wood, S., & Cheok, L.-T. (2000). Extensible mpeg-4 textual format (xmt). In Proceedings of the 2000 ACM workshops on Multimedia, MULTIMEDIA '00, (pp. 71–74)., New York, NY, USA. ACM. 14

Klyne, G., Reynolds, F., Woodrow, C., Ohto, H., Hjelm, J., Butler, M. H., & Tran, L. (2004). Composite capability/preference profiles (CC/PP): Structure and vocabularies 1.0, W3C recommendation. http://www.w3.org/TR/CCPP-struct-vocab/. 17, 18

Kozen, D. (1983). Results on the propositional μ-calculus. Theoretical Computer Science, 27, 333–354. 43, 65

Kozen, D. (1997). Kleene algebra with tests. ACM Transactions on Programming Languages and Systems (TOPLAS), 19, 427–443. 72

Kupferman, O. & Vardi, M. (1999). The weakness of self-complementation. In Proc. 16th Symp. on Theoretical Aspects of Computer Science, volume 1563 of LNCS, (pp. 455–466). 44

Laborie, S., Euzenat, J., & Layaïda, N. (2011). Semantic adaptation of multimedia documents. Multimedia Tools and Applications, 55(3), 379–398. 18

Lämmel, R. & Meijer, E. (2007). Revealing the x/o impedance mismatch. In R. Backhouse, J. Gibbons, R. Hinze, & J. Jeuring (Eds.), Datatype-Generic Programming, volume 4719 of Lecture Notes in Computer Science (pp. 285–367). Springer Berlin / Heidelberg. 4

Lassila, O. & Swick, R. R. (1999). Resource description framework (RDF). model and syntax specification, W3C recommendation. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/. 17

Layaïda, N. (1997). Madeus : système d'édition et de présentation de documents structurés multimédia. Thèse de doctorat, Université Joseph-Fourier - Grenoble I. 9, 12

Layaïda, N. (2005). SMIL 2.1 language profile. W3C recommendation, World Wide Web Consortium. 13

Layaïda, N. & Genevès, P. (2010). Debugging standard document formats. In WWW 2010: Proceedings of the 19th International Conference on World Wide Web, (pp. 1269–1272). ACM. 22

Layaïda, N. & Karmouch, A. (1998). SMIL: The world wide web standard of the W3C. In CCBR'98: Proceedings of the 2nd Canadian Conference on Broadband Research. IEEE. 13

Layaïda, N., Lemlouma, T., & Quint, V. (2005). NAC, une architecture pour l'adaptation multimédia sur le web. Technique et Science Informatiques, 24(7), 789–813. 17

Layaïda, N. & Ossenbruggen, J. V. (2001). SMIL 2.0 language profile. W3C recommendation, World Wide Web Consortium. 2, 13

Layaïda, N., Roisin, C., & Sabry-Ismaïl, L. (2001). Systèmes Multimédias Communicants, chapter Chapitre 6 : Support d'exécution de documents multimédia, (pp. 197–222). Paris, France: Hermès. 14

Layaïda, N., Sabry-Ismaïl, L., & Roisin, C. (2002). Dealing with uncertain durations in synchronized multimedia presentations. Multimedia Tools and Applications, 18(3), 213–231. 14

Lemlouma, T. & Layaïda, N. (2001). The negotiation of multimedia content services in heterogeneous environments. In MMM 2001: Proceedings of the 8th International Conference on Multimedia Modeling, (pp. 187–206). CWI. 17

Lemlouma, T. & Layaïda, N. (2003a). Adapted content delivery for different contexts. In SAINT 2003: Proceedings of the International Symposium on Applications and the Internet, (pp. 190–197). IEEE. 17

Lemlouma, T. & Layaïda, N. (2003b). Media resources adaptation for limited devices. In ELPUB 2003: Proceedings of the 7th ICCC/IFIP International Conference on Electronic Publishing, (pp. 209–218). 17

Lemlouma, T. & Layaïda, N. (2004). Context-aware adaptation for mobile devices. In MDM 2004: Proceedings of the 5th IEEE International Conference on Mobile Data Management, (pp. 106–111). IEEE. 17

Libkin, L. & Sirangelo, C. (2008). Reasoning about xml with temporal logics and automata. In LPAR '08: Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, (pp. 97–112)., Berlin, Heidelberg. Springer-Verlag. 45

Libkin, L. & Sirangelo, C. (2010). Reasoning about xml with temporal logics and automata. J. Applied Logic, 8(2), 210–232. 45, 46

Marx, M. (2004). Conditional XPath, the first order complete XPath dialect. In PODS '04: Proceedings of the twenty-third ACM Symposium on Principles of Database Systems, (pp. 13–22)., Paris, France. ACM Press. 22, 25, 148

Miklau, G. & Suciu, D. (2004). Containment and equivalence for a fragment of XPath. Journal of the ACM, 51(1), 2–45. 74

Møller, A. & Schwartzbach, M. I. (2005). The design space of type checkers for XML transformation languages. In Proc. Tenth International Conference on Database Theory, ICDT '05, volume 3363 of LNCS, (pp. 17–36). Springer-Verlag. 122

Moon, H. J., Curino, C. A., Deutsch, A., & Hou, C.-Y. (2008). Managing and querying transaction-time databases under schema evolution. In VLDB '08, (pp. 882–895). VLDB Endowment. 121

Moro, M. M., Malaika, S., & Lim, L. (2007). Preserving xml queries during schema evolution. In WWW '07, (pp. 1341–1342). ACM. 121

Murata, M., Lee, D., Mani, M., & Kawaguchi, K. (2005). Taxonomy of XML schema languages using formal language theory. ACM Transactions on Internet Technology, 5(4), 660–704. 22, 88, 94, 97, 112

Olteanu, D., Meuss, H., Furche, T., & Bry, F. (2002). XPath: Looking forward. In EDBT '02: Proceedings of the Worshop on XML-Based Data Management, volume 2490 of LNCS, (pp. 109–127). Springer-Verlag. 85

OMA (2001a). Wireless application protocol architecture specification (WAP) – open mobile alliance. Standard. 16

OMA (2001b). Wireless markup language (WML) version 2 specification – open mobile alliance. Standard. 16

Pan, G., Sattler, U., & Vardi, M. Y. (2006). BDD-based decision procedures for the modal logic K. Journal of Applied Non-classical Logics, 16(1-2), 169–208. 44, 56, 68

Pietriga, E. (2005). MathML content2presentation transformation. http://www.lri.fr/~pietriga/mathmlc2p/mathmlc2p.html. 118

Rabin, M. (1969). Decidability of Second-Order Theories and Automata on Infinite Trees. TAMS, 141, 1–35. 25

Reynolds, J. C. (1983). Types, abstraction and parametric polymorphism. In IFIP Congress, (pp. 513–523). 135

Rose, K. H. (2004). The XML world view. In DocEng '04: Proceedings of the 2004 ACM symposium on Document engineering, (pp. 34–34)., New York, NY, USA. ACM. 121

Safra, S. (1988). On the complexity of omega -automata. In Proceedings of the 29th Annual Symposium on Foundations of Computer Science, (pp. 319–327)., Washington, DC, USA. IEEE Computer Society. 44

Schmitz, P. & Cohen, A. (2001). SMIL animation, W3C recommendation. http://www.w3.org/TR/smil-animation/. 10, 14

Schmitz, P., Yu, J., & Layaïda, N. (1999). Synchronized multimedia integration language (SMIL) document object model. W3C note, World Wide Web Consortium. 13

Sedlar, E. (2005). Managing structure in bits & pieces: the killer use case for XML. In SIGMOD '05, (pp. 818–821). ACM. 121

Smith, K. (2010). Device independent authoring language (DIAL), W3C working group note. http://www.w3.org/TR/dial/. 18

Tanabe, Y., Takahashi, K., & Hagiya, M. (2008). A decision procedure for alternation-free modal μ-calculi. In Advances in Modal Logic, (pp. 341–362). 44, 45

Tanabe, Y., Takahashi, K., Yamamoto, M., Tozawa, A., & Hagiya, M. (2005). A decision procedure for the alternation-free two-way modal μ-calculus. In TABLEAUX '05: Proceedings of the 14th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, volume 3702 of LNCS, (pp. 277–291)., Koblenz, Germany. Springer-Verlag. 43, 44, 45

Thomas, W. (1990). Automata on infinite objects. In Handbook of theoretical computer science (vol. B): formal models and semantics (pp. 133–191). Cambridge, MA, USA: MIT Press. 99

Tozawa, A. (2004). On binary tree logic for XML and its satisfiability test. In PPL '04: Informal Proceedings of the Sixth JSSST Workshop on Programming and Programming Languages. 45

Vardi, M. Y. (1998). Reasoning about the past with two-way automata. In ICALP '98: Proceedings of the 25th International Colloquium on Automata, Languages and Programming, (pp. 628–641). Springer-Verlag. 44

Villard, L. (2002). Modèles de documents pour l'édition et l'adaptation de présentations multimédias. Phd thesis, Institut National Polytechnique de Grenoble. 12, 20, 21

Villard, L. & Layaïda, N. (2002). An incremental XSLT transformation processor for XML document manipulation. In WWW 2002: Proceedings of the 11th International Conference on World Wide Web, (pp. 474–485). ACM. 19

Villard, L., Roisin, C., & Layaïda, N. (2000). An XML-based multimedia document processing model for content adaptation. In DDEP 2000:

Proceedings of 8th International Conference on Digital Documents and Electronic Publishing, volume 2023 of Lecture Notes in Computer Science, (pp. 104–119). Springer. 12, 13, 19

Vion-Dury, J.-Y. & Layaïda, N. (2003). Containment of XPath expressions: an inference and rewriting based approach. In Proceedings of the Extreme Markup Languages Conference. IDEAlliance. 21

Voss, J. (2007). Wikipedia dtd. http://meta.wikimedia.org/wiki/Wikipedia_DTD. 90

Vouillon, J. (2006). Polymorphic regular tree types and patterns. In Proceedings of the 33rd symposium on principles of programming languages (POPL '06), (pp. 103–114)., Charleston, SC, USA. 141, 143

Wadler, P. (1989). Theorems for free! In Proceedings of the 4th international conference on functional programming languages and computer architecture (FPCA '89), (pp. 347–359)., London. 138

Wadler, P. (2000). Two semantics for XPath. Internal Technical Note of the W3C XSL Working Group, http://homepages.inf.ed.ac.uk/wadler/papers/xpath-semantics/xpath-semantics.pdf. 98

Walsh, N. & Muellner, L. (1999). DocBook: The Definitive Guide (1st ed.). Sebastopol, CA, USA: O'Reilly & Associates, Inc. 9, 10, 11, 20

XQDT (2010). XQDT : XQuery development tools. http://www.xqdt.org. 75

Yu, C. & Popa, L. (2005). Semantic adaptation of schema mappings when schemas evolve. In VLDB '05, (pp. 1006–1017). VLDB Endowment. 121

Zee, K., Kuncak, V., & Rinard, M. C. (2008). Full functional verification of linked data structures. In Gupta, R. & Amarasinghe, S. P. (Eds.), PLDI, (pp. 349–361). ACM. 44