Logique de requêtes à la XPath : systèmes de preuve et pertinence pratique

Anthony Lick

université PARIS-SACLAY

Thèse de doctorat ED STIC, NNT 2019SACLN016

école normale supérieure paris–saclay

XPath-like Query Logics: Proof Systems and Real-World Applicability

Thèse de doctorat
préparée à l'École Normale Supérieure Paris-Saclay
au sein du Laboratoire Spécification & Vérification

Présentée et soutenue à Cachan, le 8 juillet 2019, par

Anthony Lick

Composition du jury :

Joachim Niehren Professeur, INRIA Lille	Rapporteur
Alwen Tiu Senior Lecturer, Australian National University	Rapporteur
Véronique Benzaken Professor, Université Paris-Sud	Examinatrice
Diego Figueira Chargé de Recherche, CNRS	Examinateur
Andrzej Indrzejczak Professor, University of Łódź	Examinateur
Sara Negri Professor, University of Helsinki	Présidente du Jury
David Baelde Maître de Conférences, ENS Paris-Saclay	Co-encadrant de thèse
Sylvain Schmitz Maître de Conférences, ENS Paris-Saclay	Directeur de thèse

XPath-like Query Logics: Proof Systems and Real-World Applicability

Doctoral Thesis

Anthony Lick

Abstract

Motivated by applications ranging from XML processing to runtime verification of programs, many logics on data trees and data streams have been developed in the literature. These offer different trade-offs between expressiveness and computational complexity; their satisfiability problem has often non-elementary complexity or is even undecidable. Moreover, their study through model-theoretic or automata-theoretic approaches can be computationally impractical or lacking modularity.

In a first part, we investigate the use of proof systems as a modular way to solve the satisfiability problem of data logics on linear structures. For each logic we consider, we develop a sound and complete hypersequent calculus and describe an optimal proof search strategy yielding an NP decision procedure. In particular, we exhibit an NP-complete fragment of the tense logic over data ordinals—the full logic being undecidable—, which is exactly as expressive as the two-variable fragment of the first-order logic on data ordinals.

In a second part, we run an empirical study of the main decidable XPath-like logics proposed in the literature. We present a benchmark we developed to that end, and examine how these logics could be extended to capture more real-world queries without impacting the complexity of their satisfiability problem. Finally, we discuss the results we gathered from our benchmark, and identify which new features should be supported in order to increase the practical coverage of these logics.

Acknowledgements [*]

First of all, I would like to thank my two PhD advisors, David Baelde and Sylvain Schmitz, for being the best advisors one could hope for. These three years would not have been the same without all your precious advices and support. It has been a pleasure working with you!

I would also like to thank the reviewers of my thesis, Joachim Niehren and Alwen Tiu, for their valuable comments and thorough proofreading, especially given the tight deadline we asked you to meet. I also thank the other members of the jury—Véronique Benzaken, Diego Figueira, Andrzej Indrzejczak, and Sara Negri—for accepting to evaluate my work.

More generally, I say thank you to every member of LSV for making this lab such an amazing place. This lab really has a unique atmosphere, and I wish you all the best for your new adventures in Saclay. As for me, I will now begin a new career as a teacher in Prépa, after a first taste at the ENS Paris-Saclay. In that regard, I thank everyone who teaches at the département d'informatique. I especially thank Claudine Picaronny, as I gave exercise sessions in maths and computer science under her direction for the first two years. She gave me numerous advices, and I am grateful for all the teaching experience that she shared with me. Equivalently, I thank Stéphane Leroux and Philippe Schnoebelen, for whom I did the same during this last year. I really enjoyed working with you. I also thank Virginie, Imane, Pierre, Alexis, Francis and Hugues for keeping this lab running. Thank you Jean for all your fascinating stories. I thank Alain and Serge for all their quotes for the best of.

Speaking of quotes from the best of, I thank Simon for being the greatest office neighbour, even though he could not concentrate without performing some air battery (or just battery…). I also thank Antoine—my other office neighbour—whom, I am sure, shared my pain about the air battery but was too polite to complain. I say thank you to all the PhD students, for the laughs and discussions we shared during lunch, gouter, coffee breaks or sometimes dinner. I thank Alessio for our shared adventures in India. And I thank his office mate Adrien, even if I am disappointed he failed his duty to show OSS 117 to Alessio.

I thank Poli, Marco, Biatch, Papy and Vincent for the tentative tennis sessions which lasted for a solid six weeks, and for the beer sessions which lasted longer. I also thank them, along with Grosxor, Kulu, Gac, Edwin, Meliss, Mathilde, and Aurianne, for all the "potatoes and cheese" holidays. I thank Nobody for buying me food every time he needed me to reinstall Linux on his computer.

I now thank everybody who should have found the hidden word on this page. Président & Présidente, for hosting the upcoming party at their place, for all the movie nights, and all the dog walks. Samy & Sarah for all the cinema sessions, and the game nights on the Switch. Joris for his pétanque balls. Romaing for his improbable adventures. I thank Davy, Doc, Japan Boy, Skippy.

[*]Work funded by ANR-14-CE28-0005 PRODAQ.

I thank Alpha & Écho for helping me writing this thesis, and I thank Isaac for creating a diversion at the maternity in the meantime. Thank you Aloa for warming my lap, Yang for drooling a bit on me, and Miette for the cuddles and the scars. Thank you Caramelle for always wanting more belly rubs.

I thank my family, all those who could came to my defence, and all those who could not. I thank my parents and my brother for always supporting me, and for helping organise the pot!

Finally, thank you StackOverflow, for always being there for me.

Contents

CHAPTER 1

Introduction

Trees and linear structures are widely studied in Computer Science, as they can abstract various situations where a hierarchy or an order exist between objects. In such structures, every position often carries a label from a finite alphabet, but it is sometimes necessary to also work with *data* from an infinite domain, such as the set of integers or the set of strings over a finite alphabet.

1.1. Structures with Data

When working with databases, one typically needs to work with an infinite domain. The most standard model is the relational model introduced by Codd [1970], where relations between data are represented as tables—as illustrated in Figure 1.1—and for which query languages such as *SQL* have been designed to extract information from them. To date, this model is still used in countless applications.

Title	Author	Year
Harry Potter and the Philosopher's Stone	J.K. Rowling	1997
A Game of Thrones	George R.R. Martin	1996
The Da Vinci Code	Dan Brown	2003
⋮		

FIGURE 1.1. Example of relational database.

On the other hand, semi-structured data do not obey this formalism and rather focus on the hierarchies or relationships between data. Many shapes of semi-structured data have been studied, such as data graphs, data trees, or data words. In this thesis, we focus on trees and words with data.

1.1.1. Data Trees. The *eXtended Markup Language* (XML), introduced in the late 1990s by xml [2008], represents data in a hierarchical way. XML documents are the standard format to share data over the Internet, and consist of text files that are both human- and machine-readable. They can also be used as databases, for instance in the eXist-db[1] project.

The underlying structure of such documents is a *data tree*, which is a finite tree where every position is labelled by a letter from a finite alphabet and carries a data value from some infinite domain such as the set of strings. For example, the XML document from Figure 1.2 can be represented as shown in Figure 1.3. Data trees are formally defined in Section 6.2.1, and their link to XML documents is discussed in Section 6.2.6.

[1] `https://github.com/eXist-db`

```
<library>
  <book title="Harry Potter and the Philosopher's Stone"
    author="J.K. Rowling" year="1997"/>
  <book  title="A Game of Thrones"
    author="George R.R. Martin" year="1996"/>
  <book title="The Da Vinci Code"
    author="Dan Brown" year="2003"/>
  ...
</library>
```

FIGURE 1.2. Example of an XML document.

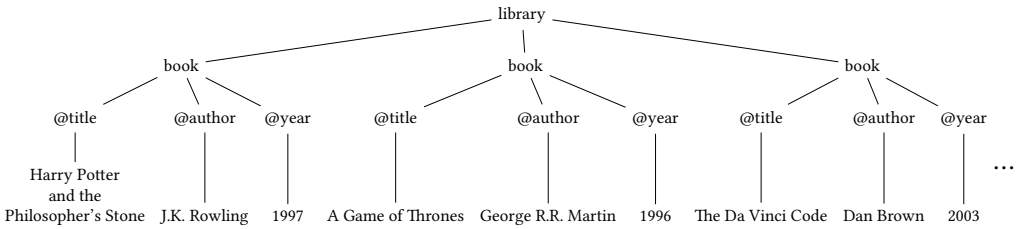

FIGURE 1.3. Representation of the XML document from Figure 1.2 as a data tree.

Just as for relational databases, one might want to extract informations from such documents. The XPath language [xpa, 1999, 2014] is arguably the most popular querying language for selecting elements in XML documents. It is embedded in the XML processing languages XSLT [xsl, 2017] and XQuery [xqu, 2014a], and widely used in general-purpose languages like Java or C# through third-party libraries. An XPath query can navigate inside the tree structure of an XML document, and perform some tests on data to extract parts of the document.

EXAMPLE 1.1. The following informations can be extracted from the tree of Figure 1.3 by some XPath queries, as we will see in Section 1.2:

(1) Is there a book titled "A Game of Thrones"?
(2) Who is its author?
(3) Which books are written by the same author as "A Game of Thrones"?
(4) Are there two books written by the same author?

1.1.2. Data Words and Ordinals. Many applications can generate *data streams*, such as traces of a program's execution [e.g. Grigore et al., 2013], system logs [e.g. Bollig, 2011], XML streams [e.g. Gauwin et al., 2011], or detecting intrusions [e.g. Goubault-Larrecq and Olivain, 2014], which motivated the study of *data words* and *data ω-words* in order to be able to formally reason about such streams. Being able to work with ordinals bigger than ω can also be highly convenient. For instance, Godefroid and Wolper [1994] rely on such ordinals to do model checking of the concurrent execution of n events while avoiding the cost of exploring the $n!$ interleavings of these events. Demri and Nowak [2007] also use such ordinals to model Zeno behaviours of physical systems.

$$(b,1)(r,1)(b,2)(r,2)(w,1)(w,2)(e,2)(e,1)(s,0)(b,3)(r,3)(e,3)(s,0)$$

FIGURE 1.4. A data word representing the logs of concurrent processes.

EXAMPLE 1.2. Consider a system where multiple processes could be editing the same file on some server. We would like to verify some properties about their concurrent execution. The log of their execution can be represented as a data word, the datum being an integer identifying the process, and the label representing their action: b for the beginning of a process, e for its ending, and r (resp. w) when a process reads (resp. writes) the file. In addition, some other process could sometimes shutdown the server (s). An example of such a log is given in Figure 1.4. On such a data word, we could want to verify various properties:

(1) Every process eventually terminates: for every position labelled by b, there is a later position with the same datum and labelled by e.
(2) More generally, we could want that for every datum, the corresponding subword belongs to $s^* + b(r+w)^*e$, i.e. every process does not do anything after it stops or before it starts.
(3) No process is interrupted by a shutdown: for every position labelled by b, there is no position labelled by s before its corresponding e.
(4) For every position labelled by w, there exists a previous position labelled by r with the same datum such that there is not a position in between labelled by w and carrying a different datum.

On the data word from Figure 1.4, all the conditions but the last one are respected.

1.2. Data Logics

1.2.1. The Satisfiability Problem. Alongside the *evaluation* of the set of elements selected in a document [e.g. Arroyuelo et al., 2015; Benedikt and Koch, 2009; Bojańczyk and Parys, 2011; Gottlob et al., 2005], the main computational problem associated with a query is the *satisfiability problem*: that is, given an XPath query, whether there is an XML document from which it will select some information. The study of XPath have been abstracted by various logics on trees or data trees, and this motivated the investigation of the satisfiability problem of such logics, which is the problem of determining if there exists a model of a given formula from the logic at hand.

SATISFIABILITY PROBLEM	
Fixed:	A logic L.
Input:	A formula φ from L.
Question:	Is there a model $\mathfrak{M}$ and a position w of $\mathfrak{M}$ such that $\mathfrak{M}, w \models \varphi$?

Studying this kind of property leads to practical optimisations, for instance by detecting dead code inside a program. This problem can even be looked at more precisely, when a *Document Type Definition* (DTD) can be provided, specifying the shape of the data trees at hand in a specific project.

Satisfiability Problem modulo DTD

Fixed: A logic L.
Input: A formula φ from L and a DTD $\mathcal{D}$.
Question: Is there a model $\mathfrak{M} \in \mathcal{D}$ and a position w of $\mathfrak{M}$ such that $\mathfrak{M}, w \models \varphi$?

For example, when doing query evaluation Groppe and Groppe [2006] first verify whether a query is satisfiable modulo a *XML Schema Definition* (XSD)—similar to a DTD—, and prevent its execution if it is not.

Other variations of the satisfiability problem have been studied, such as the *equivalence problem*—where the question is whether two formulæ have exactly the same models—, or the *entailment problem*—where the question is whether one formula implies the other.

Equivalence Problem

Fixed: A logic L.
Input: Two formulæ φ_1 and φ_2 from L.
Question: For every model $\mathfrak{M}$ and every position w of $\mathfrak{M}$, $\mathfrak{M}, w \models \varphi_1$ if, and only if, $\mathfrak{M}, w \models \varphi_2$?

These problems are closely linked to the satisfiability problem when the studied logic is closed under Boolean operators, and have also many applications. For example, when optimising queries or when updating every query of a project after changing the global structure of a database, it is natural to check if the modifications applied did not break anything by testing if the new queries are satisfiable. Furthermore, solving such problems can also involve rewriting techniques putting the input under a simpler form which could a priori also been evaluated faster, leading in turn to automatic optimisation of queries, as shown by Genevès and Vion-Dury [2004].

Similarly, the example from Section 1.1.2 illustrates what properties of data words one might want to express. For instance, we could write down formulæ corresponding to the properties at stake, and then check if they are satisfiable, i.e., if there is a data-word—abstracting some logs that could happen in practice—satisfying the formulæ. If the logic at hand is expressive enough—which is the case for many of them—we could theoretically encode the whole program in the logic as well. This would allow to do static analysis of programs, that is checking some features of the program without running it. However, writing a formula describing an entire program could prove too costly in practice, hence the most common application is *model checking*, which is similar to the satisfiability problem modulo DTD for XPath queries: given a description of the models of the problem at hand, we want to check whether some properties can be satisfied.

All these examples of applications motivate the wide study of the *satisfiability problem* of various data logics.

1.2.2. Data Logics. *Data logics* feature both a way to navigate inside a structure, and a way to compare data held at different positions.

1.2.2.1. *Data Trees.* The backbone of XPath is the Propositional Dynamic Logic on trees introduced by Afanasiev et al. [2005]. It allows to navigate inside a data tree in many different ways: from the current position, one can navigate to a child, a parent, the next or previous sibling, or use the transitive closure of one these axes. It can also perform data tests against a constant, by testing whether a path can lead to a position in the data tree with a datum matching the constant. Besides, more complex *data joins* can be done, where we can test whether two paths can lead to nodes with (non)-equal data.

Example 1.3. For instance, the following XPath queries correspond to the properties from Example 1.1:

(1) For the first property, starting from the root of the data tree, the following query first tests that the starting node is labelled 'library', then navigates to a child that must satisfy what is written between brackets: it must be labelled 'book' and have an attribute @$title$ matching 'A Game of Thrones.'

$$\texttt{self}::\text{library}/\texttt{child}::*[\text{book and } @title = \text{'A Game of Thrones'}]$$

(2) Furthermore, we can select its @$author$ attribute: the following query will return 'George R.R. Martin':

$$\texttt{self}::\text{library}/\texttt{child}::*[\text{book and } @title = \text{'A Game of Thrones'}]/@author$$

(3) Our third query must involve more complex data tests, where non-trivial paths are employed:

$$\begin{aligned}&\texttt{self}::\text{library}/\texttt{child}::*[\text{book and } @author = \\ &\quad \texttt{parent}::*/\texttt{child}::*[\text{book and } @title = \text{'A Game of Thrones'}]/@author \\ &]/@title\end{aligned}$$

This query selects books where the @$author$ attribute matches the result of the previous query (the navigational step $\texttt{parent}::*$ allows to go back to the node 'library'), and returns the @$title$ attributes of every selected book. In other words, it returns the titles of all the books written by the author of 'A Game of Thrones.'

(4) For the last property, we check if two books have the same authors, with one appearing strictly after the other for the order among siblings (this is to make sure that the two books are indeed distinct):

$$\texttt{self}::\text{library}/\texttt{child}::\text{book}[@author = \texttt{following-sibling}::\text{book}/@author]$$

When data joins and navigations in every direction are available, the satisfiability problem is undecidable for XPath queries, though various decidable fragments of XPath have been studied. The first two queries from Example 1.3 belong to Downward XPath—where data joins and only downward navigation (child and descendant) are allowed—, while the third one belongs to Vertical XPath—where upward navigation (parent and ancestor) are also allowed—, and the last one belongs to Forward XPath—which enriches Downward XPath with the following-sibling axis. All these fragments are discussed more thoroughly in Section 1.2.3, and will be the subject of a survey in Part 2.

Another natural approach is the First Order Logic as it is the bedrock of the language SQL—widely used to query relational databases. In that sense, Bojańczyk et al. [2009] thoroughly study the First Order Logic on data trees. This logic is equipped with binary predicates indicating if two variables x and y are representing nodes such that:

- y is a child of x: $E_{\downarrow}(x, y)$.
- y is the next sibling of x: $E_{\rightarrow}(x, y)$.
- x and y have the same datum: $x \sim y$.

However it is necessary to consider the two variable fragment to get a decidable logic; and adding the transitive closure of $E_{\downarrow}$ and $E_{\rightarrow}$ makes the satisfiability problem harder. In this fragment, Property 4 from Example 1.1 can also be expressed. But since only two variables can be used at once, data joins are often expressed in the following fashion:

$$\begin{aligned}
&\exists x.\ \text{library}(x) \wedge \forall y.\ \neg E_{\downarrow}(y, x) \\
\wedge\ &\exists y.\ E_{\downarrow}(x, y) \wedge \text{book}(y) \\
\wedge\ &\exists x.\ E_{\downarrow}(y, x) \wedge @author(x) \\
\wedge\ &\exists y.\ \neg(x = y) \wedge x \sim y \wedge @author(y) \\
\wedge\ &\exists x.\ E_{\downarrow}(x, y) \wedge \text{book}(x) \\
\wedge\ &\exists y.\ E_{\downarrow}(y, x) \wedge \text{library}(y) \wedge \forall x.\ \neg E_{\downarrow}(x, y)
\end{aligned}$$

This formula starts by following a first path from the root, then jumps to a node with matching datum thanks to the $\sim$ operator, and finally follows the second path backwards until it reaches the root again. Remark that we usually cannot check that we reached again the starting node at the end—as this information has been lost during the navigation—but we can always check whether a node is the root. Hence, typically only data joins with one path involving the root can be translated in this logic.

1.2.2.2. *Data Words.* The First Order Logic on data words has also been investigated, by Bojańczyk et al. [2011], where the $+1$ operator points to the next position, and the order relation $<$ allows to specify the relative positions between two variables. As for data trees, only the two variable fragment is decidable. For instance, Property 1 from Example 1.2 can be expressed by the formula

$$\forall x.\ (b(x) \Rightarrow \exists y.\ e(y) \wedge x < y \wedge x \sim y).$$

However, Property 3 does not seem expressible with only two variables, as we need to specify that the desired e position is between the b and s at hand.

Various other logics have been extended with a way to work with data. Linear Temporal Logic introduced by Pnueli [1977] is the base of the Logic of Repeating Values [Demri et al., 2012, 2016], and of Freeze LTL [Demri and Lazić, 2009; Figueira and Segoufin, 2009; Lazić, 2011] which uses the freeze quantifiers introduced by Alur and Henzinger [1994]. XPath can also be seen as a logic on data words when working with the horizontal axes only [Figueira and Segoufin, 2009], since we are working with unranked trees. In these logics, the G modality expresses that something holds in every future position, and the dual F modality expresses the existence of a future position where some property is true. In addition, the $\downarrow_r$ operator stores the datum of the current position in the register r, and the atom $\uparrow_r$ allows to test later in the formula if the current position carries the datum stored in r.

Example 1.4. For instance, Property 1 from Example 1.2 can be expressed using freeze quantifiers and tense logic by:

$$\mathsf{G}\,(b \Rightarrow \downarrow_r \mathsf{F}\,(e \wedge \uparrow_r)).$$

Furthermore, Property 3 can be expressed by:

$$\mathsf{G}\,(b \Rightarrow \downarrow_r \mathsf{G}\,(s \Rightarrow \neg \mathsf{F}\,(e \wedge \uparrow_r))).$$

Moreover, some fragments of XPath have also been studied over data words [Figueira, 2012b; Figueira and Segoufin, 2009]. Finally, the case of infinite data words has also been considered, for instance by Lazić [2011] for freeze LTL, or by Colcombet and Manuel [2014] concerning Fixpoint Logic on data words extending the modal μ-calculus.

1.2.2.3. *Techniques.* Many approaches have been developed to solve the satisfiability problem of the considered logics. Regarding tree logics, some *proof-theoretic* techniques are sometimes used, for instance by Afanasiev et al. [2005]; ten Cate and Lutz [2009] for data-free logics, or Abriola et al. [2017a] for data logics, where an axiomatisation of a logic is described, which can lead to rewriting systems for query optimisation.

A different—*model-theoretic*—approach is used for instance by Figueira [2012a], where the shape and size of models of the downward fragment of XPath are studied, to then search for a small model of a given query. However, this approach does not yield any effective algorithm, as the corresponding decision procedure consists of guessing a potential model of a given shape and test if it is actually a model of the input.

The most common approach relies on *automata-theoretic* techniques—both for the data-free case and for data logics—usually by building an enriched automaton recognising the models of a given formula and testing it for emptiness.

For instance, Vardi and Wolper [1986] capture the linear time Propositional Temporal Logic with Büchi automata; and various tree automata are used to study branching logics [Calvanese et al., 2009; ten Cate and Segoufin, 2008; Vardi, 1998].

Concerning data logics, a *data-automaton* from Bojańczyk et al. [2011] consists of a letter-to-letter transducer A and a finite automaton B. The transducer A must be run first on the input data word—checking global properties—, then B must accept every subword of the output of A induced by a datum—thus checking local properties for each datum. This idea of checking both *global* and *datum-specific* properties of a data word is also present in the *class memory automata* introduced by Björklund and Schwentick [2010], which are expressively equivalent to *data-automata*. Another type of automaton to work with data words are the *register-automata* introduced by Kaminski and Francez [1994], where a constant number of registers can store data values encountered somewhere in the input, which can then be compared to the datum carried by another position of the data word visited later in the run. This type of automata is used for instance by Demri and Lazić [2009] with alternation to study freeze LTL, or to investigate various fragments of XPath [Figueira, 2012b; Figueira and Segoufin, 2009, 2017].

Even though their techniques are similar, the automata used for one specific logic are often ad hoc for that logic, thus this approach is not really modular: each different logic requires a tailored automaton model. Furthermore, such an approach may not easily lead to an optimal procedure when studying fragments of smaller complexity for a given logic, as we will discuss in Section 2.1. In order to avoid these downsides, we investigate in Part 1 how *proof systems* could lead to algorithms of optimal complexity for such logics.

1.2.3. Fragments and Complexity. Data logics are the natural product of adding a way to compare data stored at different positions of a model to some decidable navigational logics working on the same models, but the newly obtained logics are often undecidable. In that

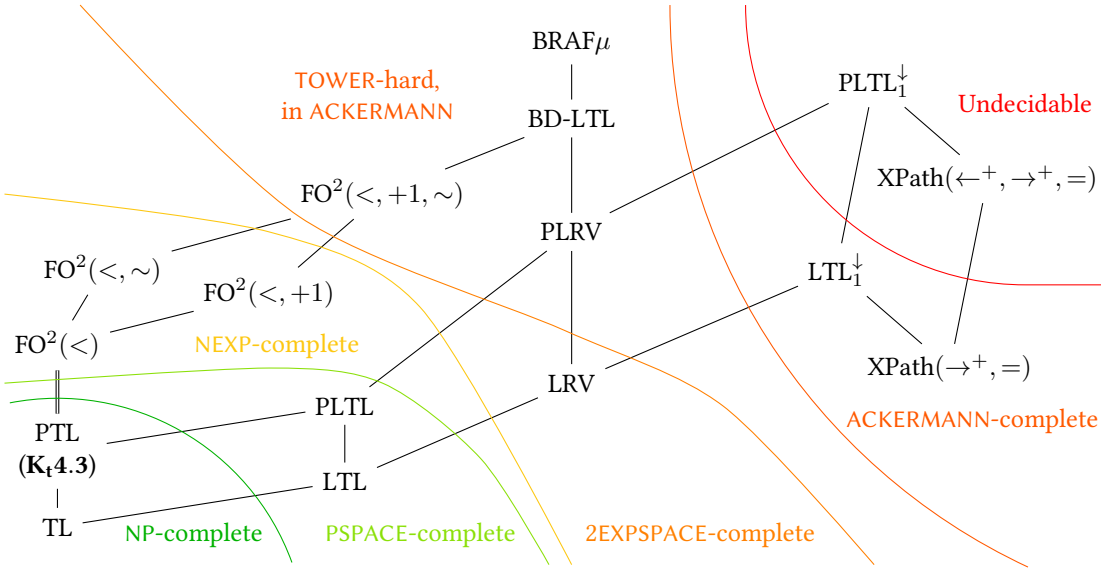

FIGURE 1.5. Inclusions and complexities of some logics over linear structures.

sense, various fragments of these logics have been studied, restricting the expressivity on the navigational side to get back the decidability of the satisfiability problem.

In the case of linear structures, the Linear Time Logic—which is PSPACE-complete [Sistla and Clarke, 1985], both with only future-looking modalities or with past modalities as well—have been extended by multiple logics to work with data. These fragments are presented below, and the complexities and inclusions between them are illustrated in Figure 1.5.

- The Logic of Repeating Values (LRV) [Demri et al., 2016], which is evaluated on linear structures with multiple attributes (when working with only one attribute, these structures coincide with data words). The logic is able to navigate to a future position with an attribute equal (resp. different) to an attribute of the current position. Demri et al. [2016] shows that the satisfiability problem of LRV is 2EXPSPACE-complete, and also consider its extended version where such navigation can also be done to the past (PLRV), for which the satisfiability problem is shown to be equivalent to the reachability problem in VASS, which is currently known to be TOWER-hard [Czerwiński et al., 2019] and in ACKERMANN [Leroux and Schmitz, 2019].
- Freeze LTL ($\mathrm{LTL}^{\downarrow}$) [Demri and Lazić, 2009], which extends LTL by freeze quantifiers à la Alur and Henzinger [1994] and is evaluated on data words. The logic can store the datum of the current position in a register, navigates inside the structure as in LTL, and test whether the datum of the current position matches the one stored in a register. Its satisfiability problem is undecidable when at least two registers are allowed. When restricted with only one register ($\mathrm{LTL}^{\downarrow}_1$), its satisfiability problem becomes ACKERMANN-complete,[2] but remains undecidable when extended with

[2]When working with infinite data words, $\mathrm{LTL}^{\downarrow}_1$ becomes undecidable, whereas the complexities of the other logics do not change.

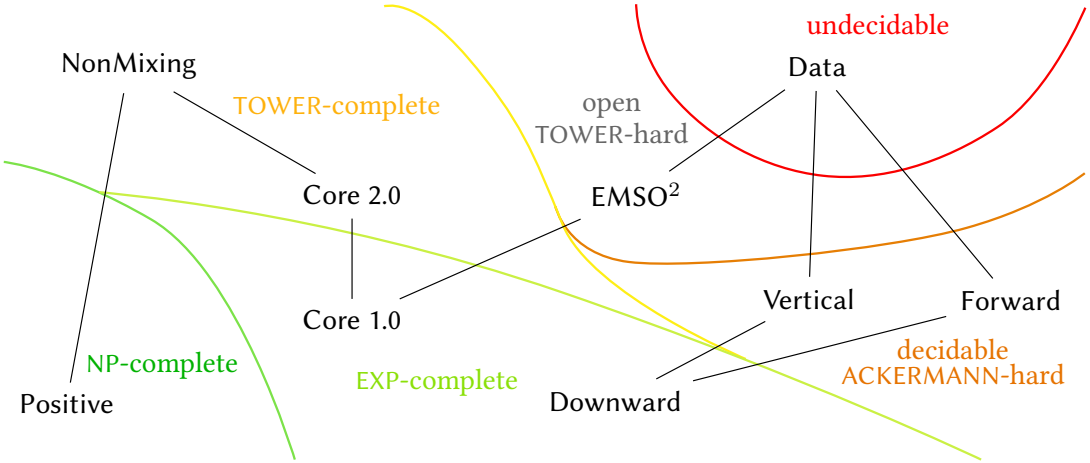

FIGURE 1.6. Inclusions and complexities of some fragments of XPath.

past navigation ($\text{PLTL}_1^{\downarrow}$). These fragments contain the corresponding Logic of Repeating Values when restricting it with only one attribute. Moreover, $\text{LTL}_1^{\downarrow}$ subsumes the XPath on words fragment $\text{XPath}(\rightarrow^+, =)$ [Figueira and Segoufin, 2009], which features data joins and can only navigate along the `following-sibling` axis, and $\text{PLTL}_1^{\downarrow}$ contains the corresponding $\text{XPath}(\leftarrow^+, \rightarrow^+, =)$ fragment with the `preceding-sibling` axis added.

- PLRV is also contained in BD-LTL from Kara et al. [2010], which is itself contained in the bounded-reversal alternation-free fragment of the μ-calculus from Colcombet and Manuel [2014]—denoted by BRAFμ on Figure 1.5.

The two variable fragment of the First Order Logic over data words has also been studied, with various signatures ranging among $<$, $+1$ and $\sim$. All of them are NEXP-complete [Bojańczyk et al., 2011; Etessami et al., 2002], except for $\text{FO}^2(<, +1, \sim)$ which is equivalent to the reachability problem in VASS [Bojańczyk et al., 2011]. For comparison, the least expressive fragment, $\text{FO}^2(<)$, is equally expressive to the Tense Logic with past navigation [Etessami et al., 2002] (as denoted by the double edge on Figure 1.5), which is the fragment of LTL with past with only the G and H modalities (and their dual F and P), and for which the satisfiability problem is NP-complete [Ono and Nakamura, 1980]; and the most expressive fragment, $\text{FO}^2(<, +1, \sim)$, is subsumed by the μ-calculus fragment from Colcombet and Manuel [2014].

Concerning XPath on trees, the complexities and relations between various fragments is summed up in Figure 1.6. The data-free fragment named XPath Core 1.0—containing navigation in every direction—is EXP-complete, while Data-XPath is undecidable. Here again decidability can be recovered by restricting the navigation allowed:

- Downward-XPath, where only navigation to a child or a descendant is possible, is EXP-complete [Figueira, 2012a].
- Forward-XPath, extending Downward-XPath with navigation to following-siblings, is decidable but ACKERMANN-hard [Figueira, 2012b; Figueira and Segoufin, 2009].

- Vertical-XPath, extending Downward-XPath with converse navigation to the parent or an ancestor, is decidable but ACKERMANN-hard [Figueira and Segoufin, 2009, 2017].

Other fragments of XPath have been studied. Jurdziński and Lazić [2007] developed a modal μ-calculus which is subsumed by ForwardXPath. The Core 2.0 fragment of XPath extends its 1.0 version, in particular by adding a way to test whether the nodes selected by two queries are the same; ten Cate and Lutz [2009] proved that this fragment is TOWER-complete. It is contained in NonMixing-XPath introduced by Czerwinski et al. [2017], where a query cannot mix equality tests and inequality tests between data, which contains XPath Core 2.0 and is still TOWER-complete. Bojańczyk et al. [2009] also investigate a fragment of XPath containing Core 1.0 that can be encoded in EMSO^2, which is TOWER-hard and for which decidability is open. On the other hand, removing negation from Data-XPath leads to an NP-complete fragment, as shown in Geerts and Fan [2005]. This illustrates that there is a trade-off to be made between expressivity and complexity when choosing a decidable fragment that will fit best one's needs. The concrete relevance of these fragments will be investigated in Part 2.

1.3. Contributions

As shown in the previous section, many logics on data words and data trees have been developed. In each case, their satisfiability problem has been studied. However, for most of them, this problem is either undecidable or non elementary, and the ones of lowest complexity are also the lowest in expressivity. It is natural to investigate this trade-off in a real-world setting, and look for other ways to increase the practical coverage of a logic without increasing its complexity.

Moreover, we saw that the study of these logics involved many different techniques, which were not always very modular: it can be difficult to adapt the techniques from one logic to another closely related logic. One may wonder whether proof-theoretic techniques could avoid this problem.

Answering these questions, the contributions of this thesis are divided in two parts.

1.3.1. Effective Proof Systems. First, we develop techniques to solve the validity problem—dual of the satisfiability problem—of some modal and data logics on linear structures via proof systems.

VALIDITY PROBLEM	
Fixed:	A logic L.
Input:	A formula φ from L.
Question:	For all models $\mathfrak{M}$ and for all positions w of $\mathfrak{M}$, does $\mathfrak{M}, w \models \varphi$?

One of the most famous kinds of proof systems is the *sequent calculus* [Gentzen, 1935], but such proof systems are often not expressive enough to capture modal logics especially with converse navigation. Therefore, we turn towards *hypersequents*—one of the various extensions of sequents—inspired by how Indrzejczak [2016] handled the future and past navigation inside linear structures.

The main benefit from using proof systems is modularity: instead of providing an ad hoc calculus for a given logic, we developed a proof system that can be easily enriched to capture

various logics, and resulting in *sound* and *complete* calculi for each of them. Moreover, we focused on having a *proof strategy* of optimal complexity in every logic we studied. More precisely:

- In Chapter 3, we investigate $\mathbf{K_t4.3}$, the tense logic over linear frames, and provide a sound and complete hypersequent calculus with optimal coNP *proof search*. This chapter corresponds to the work from Baelde et al. [2018a]. However, several improvements have been made in later works and have been incorporated in this chapter.
- In Chapter 4, we enrich the work of Chapter 3 to provide a sound and complete proof system when working over *ordinals*, i.e. *well-founded* structures. This hypersequent calculus also has optimal coNP proof search, and can be easily adapted to work with a given *order-type*. This chapter is an updated version of the work from Baelde et al. [2018b].
- In Chapter 5, we extend the previous proof system to work with freeze LTL [Demri and Lazić, 2009]. As this logic is undecidable, we exhibit a decidable fragment—equivalent to $\mathrm{FO}^2(<, \sim)$—for which our hypersequent calculus is sound and complete, and we present a *proof strategy* of optimal coNP complexity. This chapter corresponds to the work from Lick [2019].

This last proof system for logics on data ordinals could serve as a stepping stone for the study of more complex logics, such as fragments of XPath. In order to go further than the work of Baelde et al. [2016], we need to handle bidirectional modalities, which we did in the case of linear structures. Moreover, our work could be used to handle XPath navigation between siblings. However, our proof systems benefit good algorithmic properties—proof search is in coNP—, that will be lost when working with more complex logics.

1.3.2. XPath in the Real World. As shown on Figure 1.6, many decidable fragments of XPath have been studied, with various complexities and expressiveness. The goal of the second part is to investigate the practical relevance of these fragments developed for the study of XPath, and to examine what kind of extension could be reasonably made to a fragment in order to improve its coverage without impacting the complexity of its satisfiability problem.

- To that end, we developed a benchmark [Baelde et al., 2019b] by parsing XPath queries [Baelde et al., 2019c] from many open source projects and outputting their *Abstract Syntax Trees* (AST) in an XML document. We then wrote *Relax NG* schemas accepting the AST of the queries belonging to a given fragment, for each fragment and its extensions, and we measured which fragments captured the most queries. The development of these tools is presented in Chapter 6.
- In Chapter 7, we present in more detail the XPath fragment we considered, and provide the first results from our benchmark.
- In Chapter 8, we investigate what extensions can be made to these fragments without affecting the complexity of the satisfiability problem. We provide both positive and negative answers, and present the final results of our benchmark for the extended fragments.

This survey allowed to identify which fragments and which extensions should be the focus of future works about XPath. This part corresponds to the work from Baelde et al. [2019a].

Part 1

Effective Proof Systems for Tense Logic

CHAPTER 2

Preliminaries

Contents

2.1. Introduction

Solving Satisfiability. Some of the algorithmic results for the logics mentioned in Chapter 1 have been obtained via *model-theoretic* techniques (see S1.2.2.3), by showing that if a formula has a model, then it has a 'small' one, and it is actually possible to proceed similarly for the logics that we will study in this part. However, as the resulting algorithms consist essentially in guessing a model, they are impractical as they are unlikely to avoid the (high) worst case complexity of the problem. In the case of the full linear temporal logic, this has motivated the use of *automata-theoretic* techniques [Bruyère and Carton, 2007; Demri and Rabinovich, 2010; Rohde, 1997; Vardi and Wolper, 1986], typically by building an automaton of at most exponential-size recognising the set of models of the formula: checking the language non-emptiness of the automaton can then be performed on-the-fly in PSPACE and can rely in practice on a rich algorithmic toolset. However, this approach may lack modularity since the automata used to study a given logic are usually ad hoc objects specifically designed for the logic at hand. Moreover, the tense logic [Blackburn et al., 2001; Cocchiarella, 1965]—presented in Section 2.4—is an NP-complete sublogic of LTL; and it is not immediate how to tailor the approach of Vardi [1998] and Demri and Rabinovich [2010] to recover an optimal NP upper bound, because the automata for tense logic may require exponential-size. Finally, if one's interest is to check that a formula φ is valid, neither the model-theoretic nor the automata-theoretic approach yields a 'natural' certificate that could be checked by simple independent means.

Proof Theoretic Approach. All these considerations motivate our use of *proof-theoretic* techniques. In their simplest form, those can be Hilbert-style axiomatisations which, in the context of modal logic, allow to characterise valid formulæ in a way that is modular with respect to

the considered classes of models. However, these systems are not directly amenable to automated reasoning, which is rather achieved through more structured proof systems, the seminal example being Gentzen's sequent calculus.

Sequent Calculi. Our own interest in (enriched) sequent calculi, compared to e.g. axiomatisations, is that their associated proof-search procedures often yield decidability and even complexity results for the satisfiability and validity problems. Moreover, contrary for instance to automata-theoretic approaches, they are also modular, allowing them to be easily adapted to handle extensions or fragments of the logic at hand. Furthermore, their calculus rules can often be obtained from the axiomatisation of the logic. Finally, such proof systems take advantage of the link between syntax and semantics to reason only with syntactic objects.

However, basic sequent calculi are often ill-suited for modal logics, as we will see at the end on this chapter: the class of frames underlying the logic is typically difficult to capture. Therefore, more expressive variants of the sequent calculus have been developed, such as labelled sequents [Negri, 2005], display calculus [Belnap, 1982; Kracht, 1996], nested sequents [Brünnler, 2009; Kashima, 1994; Lellmann, 2015; Poggiolesi, 2009a,b], linear nested sequents [Lellmann, 2015] or hypersequents [Avron, 1991; Indrzejczak, 2012, 2015, 2016, 2017; Kurokawa, 2014]. These enriched formalisms remain quite modular and sustain extensions simply by adding a few rules. They can be exploited to provide optimal complexity solutions to the validity problem directly by proof search [Baelde et al., 2016; Das and Pous, 2017; Kanovich, 1991; Lincoln et al., 1992], which may sometimes avoid the worst-case complexity of the problem and rely in practice on various heuristics. Finally, this approach obviously yields a proof of validity as a certificate in case of success.

Contents. In this chapter, we illustrate our approach on some simple logics. In Section 2.4, we introduce the Tense Logic that will be at the heart of the logics studied in this part. The design of an effective proof system for this logic and its classical extensions will be the focus of Chapter 3. In Chapter 4, we show how to extend this calculus when working with the Tense Logic over ordinals, while still having proof search of optimal complexity. Finally, in Chapter 5, we extend the logic from Chapter 4 with freeze quantifiers à la Alur and Henzinger [1994]—as done by Demri and Lazić [2009] for LTL—, and provide new calculus rules to handle them. We then exhibit a decidable fragment of the obtained logic, along with a proof strategy leading again to a proof search algorithm of optimal coNP complexity.

2.2. Propositional Logic

We start by recalling the syntax and semantics of the Propositional Logic, and by presenting a sequent calculus à la Gentzen [1935] for that logic.

2.2.1. Syntax. Let Φ be a countable set of *propositional variables*. The *Propositional Logic* has the following syntax:

$$\varphi ::= \bot \mid p \mid \varphi \supset \varphi \qquad \text{(where } p \in \Phi\text{)}$$

The symbol $\supset$ represents the logical *implication*. Along with $\bot$ (falsum), it can express the other Boolean connectives:

$$\top = \bot \supset \bot, \quad \neg\varphi = \varphi \supset \bot, \quad \varphi \vee \psi = (\varphi \supset \bot) \supset \psi, \quad \varphi \wedge \psi = (\varphi \supset (\psi \supset \bot)) \supset \bot.$$

2.2.2. Semantics. A *valuation* is a function $V : \Phi \to \{0, 1\}$ mapping each variable from Φ to a Boolean. Given such a function, we define the *satisfaction* relation $V \models \varphi$, where φ is a formula, by structural induction on φ:

$$
\begin{array}{lll}
V \not\models \bot & & \\
V \models p & \text{iff} & V(p) = 1 \\
V \models \varphi \supset \psi & \text{iff} & \text{if } V \models \varphi \text{ then } V \models \psi
\end{array}
$$

2.2.3. Sequent Calculus. The propositional satisfiability problem (SAT) has been the first problem to be proved NP-complete [Cook, 1971]. This problem, along with its dual validity problem, have been thoroughly studied, as the development of efficient and scalable SAT solvers in the last decade illustrates [Malik and Zhang, 2009]—this is often referenced as the "SAT revolution" [Vardi, 2014]. To that end, many proof-theoretic approaches have been designed. One of the most famous proof systems is the *sequent calculus*. It manipulates *sequents*, which are syntactic objects containing formulæ, and on which some rules can be applied.

Definition 2.1. A *sequent* (denoted S) is a pair of two finite sets of formulæ, written $\Gamma \vdash \Delta$. It is satisfied by a valuation V if, whenever all the formulæ of Γ are satisfied, at least one formula of Δ if also satisfied. In that case, we write $V \models \Gamma \vdash \Delta$.

Following that definition, a sequent is *valid* when any valuation satisfies it.

There exist many versions of sequent calculi for the Propositional Logic, and we consider here a *cut-free* and *weakening-free* system, as it contains only few rules and is well adapted to *proof search*. The rules of this sequent calculus are presented in Figure 2.1.

$$
\frac{}{\varphi, \Gamma \vdash \Delta, \varphi}\ (\mathsf{ax}) \qquad \frac{}{\Gamma, \bot \vdash \Delta}\ (\bot)
$$

$$
\frac{\varphi \supset \psi, \Gamma \vdash \Delta, \varphi \quad \varphi \supset \psi, \psi, \Gamma \vdash \Delta}{\varphi \supset \psi, \Gamma \vdash \Delta}\ (\supset \vdash) \qquad \frac{\varphi, \Gamma \vdash \Delta, \psi, \varphi \supset \psi}{\Gamma \vdash \Delta, \varphi \supset \psi}\ (\vdash \supset)
$$

Figure 2.1. The rules of the propositional sequent calculus.

A rule from Figure 2.1 allows to derive a sequent conclusion from a finite number of premises. Every rule has an *active formula* in its conclusion. In every example, such formulæ will be displayed in orange. A sequent S is *provable* if it has a derivation in the system from Figure 2.1, i.e., if there exists a finite *proof tree* such that:

- the root of the tree is S;
- every internal node of the tree is a sequent whose children correspond to premises of a rule that can be applied on that sequent;
- every leaf is closed by either (ax) or ($\bot$).

Example 2.2 (Peirce's Law). For instance, the formula $\varphi = ((p \supset q) \supset p) \supset p$ is provable, as shown in Figure 2.2.

$$\dfrac{\dfrac{\dfrac{}{(p \supset q) \supset p, p \vdash p, \varphi, q, p \supset q}\,(\mathsf{ax})}{(p \supset q) \supset p \vdash p, \varphi, p \supset q}\,(\vdash \supset) \qquad \dfrac{}{(p \supset q) \supset p, p \vdash p, \varphi}\,(\mathsf{ax})}{\dfrac{(p \supset q) \supset p \vdash p, \varphi}{\vdash ((p \supset q) \supset p) \supset p}\,(\vdash \supset)}\,(\supset \vdash)$$

FIGURE 2.2. Proof of Peirce's Law from Example 2.2 in the sequent calculus from Figure 2.1.

EXAMPLE 2.3. Other rules are classically used to handle the other Boolean connectives. For instance, the following rule can be applied on a disjunction on the left-hand side of the turnstile:

$$\dfrac{\Gamma, \varphi \vdash \Delta \quad \Gamma, \psi \vdash \Delta}{\Gamma, \varphi \vee \psi \vdash \Delta}\,(\vee \vdash)$$

This rule is admissible in our calculus, as it corresponds to the following inference on the formula $(\varphi \supset \bot) \supset \psi$ (where some useless formulæ has been discarded):

$$\dfrac{\dfrac{\Gamma, \varphi \vdash \Delta}{\Gamma \vdash \varphi \supset \bot, \Delta}\,(\vdash \supset) \qquad \Gamma, \psi \vdash \Delta}{\Gamma, (\varphi \supset \bot) \supset \psi \vdash \Delta}\,(\supset \vdash)$$

2.2.4. Soundness and Completeness. The main natural question about a proof system is the link between provable objects and valid objects. We say that a sequent calculus is *sound* with respect to a logic L if all provable sequents are valid in L; and that it is *complete* with respect to L if all L-valid sequents are provable.

The completeness of a proof system can be established in various ways. Depending on how it is proved, some algorithmic results can be derived. For instance, the following properties suffice to prove the completeness of our sequent calculus:

finite branch: there is a *proof strategy* ensuring that any proof attempt will generate a finite partial proof tree, in which some leaves may be unjustified but on which the proof strategy does not allow any additional derivation steps. Such sequents on which the proof strategy is stuck are called *failure sequents*.

failure: all failure sequents are invalid.

invertibility: all the rules are invertible, i.e., for any instance of a deduction rule where the conclusion hypersequent is valid, all premisses are also valid.

The finite branch property ensures that proof search always terminates, and will reach a failure sequent in finitely many steps if the input is not provable. In such a case, the failure property establishes that the proof search actually reached an invalid sequent, and the invertibility property allows to deduce that every sequent along that branch of the partial proof tree are invalid; hence, in particular, that the input—at the root—is invalid. Moreover, the finite branch property can be refined to obtain effective complexity bounds from this proof of completeness.

For instance, a trivial proof strategy for the sequent calculus from Figure 2.1 is to forbid a rule application if one of the premises would be equal to its conclusion. For this proof strategy, all branches of a proof attempt are of polynomial length because the calculus has the subformula property. Since the invertibility property ensures no backtracking is needed during proof

search, this gives an optimal coNP proof search algorithm for solving the propositional validity problem.

Example 2.4. For instance, the formula $(p \supset q) \supset q$ is not provable in our sequent calculus. Proof search on this formula will lead to this partial proof tree, reaching the failure sequent $p \supset q \vdash p, q, (p \supset q) \supset q$. When reaching such a failure sequent, a counter-model is simply obtained by taking the valuation where an atom is true if it appears on the left-hand side of the turnstile, and false otherwise. In that case, the valuation $V(p) = V(q) = 0$ is a counter-model of the failure sequent, and therefore of the original formula by invertibility.

$$\dfrac{\dfrac{p \supset q \vdash p, q, (p \supset q) \supset q \quad \overline{p \supset q, q \vdash q, (p \supset q) \supset q}\ \text{(ax)}}{p \supset q \vdash q, (p \supset q) \supset q}\ (\supset \vdash)}{\vdash (p \supset q) \supset q}\ (\vdash \supset)$$

We will follow the same approach to prove completeness of the other proof systems presented in this part, as it will also justify that the provided proof strategies entail optimal proof search algorithms.

2.3. Future Looking Logic on Linear Frames

Modal logics are expressive and intuitive languages for describing properties of relational structures. Accordingly, when investigating properties of linear frames, it is often quite useful to express them using a tense logic [Prior, 1957] able to reason on temporal flows. For instance, LTL [Pnueli, 1977; Sistla and Clarke, 1985] and CTL [Clarke and Emerson, 1981] extend tense logic and are widely used for verifying computer programs.

In such a logic, the Propositional Logic is extended with *modal operators*. Then, a modal formula is not evaluated against just a valuation, but against a relational structure in which modalities allow to navigate. Such a model is called a *Kripke* structure, consisting of a set of *worlds*, each having its own valuation, and relations between these worlds associated with the different modalities of our logic.

For instance, the modal logic **K4.3** works on linear structures, and has a modality G quantifying over future worlds.

2.3.1. Syntax.

More precisely, the syntax of Propositional Logic is extended as follows:

$$\varphi ::= \bot \mid p \mid \varphi \supset \varphi \mid \mathsf{G}\,\varphi \qquad \text{(where } p \in \Phi\text{)}$$

Formulæ $\mathsf{G}\,\varphi$ are called *modal formulæ*. Intuitively, $\mathsf{G}\,\varphi$ expresses that φ holds 'globally' in all future worlds reachable from the current one. Moreover, we define, as is common, $\mathsf{F}\,\varphi = \neg\mathsf{G}\,\neg\varphi$ expressing that φ will hold 'in the future.'

2.3.2. Semantics.

A *frame* is a pair $\mathfrak{F} = (W, \mathcal{R})$, where W is a set of worlds, and $\mathcal{R} \subseteq W \times W$ is a binary relation over W. A *Kripke structure* is a pair $\mathfrak{M} = (\mathfrak{F}, V)$, where $\mathfrak{F} = (W, \mathcal{R})$ is a frame, and $V : \Phi \to 2^W$ is a valuation function. Given such a structure, we define the

satisfaction relation $\mathfrak{M}, w \models \varphi$, where $w \in W$ and φ is a formula, by structural induction on φ:

$$\begin{array}{lll} \mathfrak{M}, w \not\models \bot & & \\ \mathfrak{M}, w \models p & \text{iff} & w \in V(p) \\ \mathfrak{M}, w \models \varphi \supset \psi & \text{iff} & \text{if } \mathfrak{M}, w \models \varphi \text{ then } \mathfrak{M}, w \models \psi \\ \mathfrak{M}, w \models \mathsf{G}\,\varphi & \text{iff} & \forall w' \in W \text{ such that } w \mathcal{R} w', \mathfrak{M}, w' \models \varphi \end{array}$$

When $\mathfrak{M}, w \models \varphi$, we say that $(\mathfrak{M}, w)$ is a *model* of φ.

A formula that is satisfied in all worlds of all structures is said to be *valid*. In this part, we shall not consider the validity problem in general, but only in restricted classes of structures. Namely, we will consider the logic of *linear frames*, i.e., the formulæ that hold in all structures whose accessibility relation is transitive and non-branching.

This logic can be defined axiomatically, as the set of theorems generated by necessitation, modus ponens and substitution from classical tautologies and the axioms [Blackburn et al., 2001, p. 195]:

$$\mathsf{G}\,(p \supset q) \supset (\mathsf{G}\,p \supset \mathsf{G}\,q) \tag{\textbf{K}}$$

$$\mathsf{F}\,\mathsf{F}\,p \supset \mathsf{F}\,p \tag{\textbf{4}}$$

$$\mathsf{F}\,p \wedge \mathsf{F}\,q \supset \mathsf{F}\,(p \wedge \mathsf{F}\,q) \vee \mathsf{F}\,(p \wedge q) \vee \mathsf{F}\,(q \wedge \mathsf{F}\,p) \tag{.\textbf{3}}$$

The first axiom **K** is simply the Kripke schema. The next axiom, dubbed **4**, corresponds to the transitivity of $\mathcal{R}$. More precisely, canonical models of **4** are transitive [Blackburn et al., 2001]. Similarly, canonical models of the *trichotomy* axiom .**3** have accessibility relationships that are non-branching to the right.

2.3.3. Sequent Calculus. The propositional sequent calculus from Figure 2.1 can be extended with the rule from Figure 2.3 dealing with G formulæ. In this rule, Γ, Γ', Δ are sets of formulæ, and $\mathsf{G}\,[\Gamma] = \{\mathsf{G}\,\varphi \mid \varphi \in \Gamma\}$.

$$\frac{\left(\Gamma, \mathsf{G}\,[\Gamma] \vdash (\varphi_i)_{i \in I}, (\mathsf{G}\,\varphi_j)_{j \notin I}\right)_{\emptyset \neq I \subseteq \{1,\ldots,n\}}}{\mathsf{G}\,[\Gamma], \Gamma' \vdash \mathsf{G}\,\varphi_1, \ldots, \mathsf{G}\,\varphi_n, \Delta}\ (\mathsf{G})$$

FIGURE 2.3. The modal rule extending our sequent calculus.

To better understand this rule, let us convert the axiom .**3** with G modalities:

$$\mathsf{G}\,(p \vee \mathsf{G}\,q) \wedge \mathsf{G}\,(p \vee q) \wedge \mathsf{G}\,(q \vee \mathsf{G}\,p) \supset \mathsf{G}\,p \vee \mathsf{G}\,q.$$

Soundness. Generalized to n propositional variables, we obtain something of the shape of the rule from Figure 2.3. Intuitively, when working with the propositional rules from Figure 2.1, the proof search is working on a fixed world inside a structure, and when using the rule (G), the proof search jumps to a future world of the structure. Hence, since our models are transitive, the premises can keep modal formulæ from $\mathsf{G}\,[\Gamma]$ on the left-hand side of the turnstile, as well as their subformulæ from Γ, because all these formulæ can be assumed in a future position if formulæ from $\mathsf{G}\,[\Gamma]$ were assumed on the current position. Moreover, non-modal formulæ from Γ' and Δ must be discarded, as we have no information about whether they hold in the future. There is no need to remember these formulæ since this logic cannot go back to the past.

$$\dfrac{\dfrac{\dfrac{}{p \vdash p}\,(\text{ax}) \quad \dfrac{}{\mathrm{G}\,q \vdash \mathrm{G}\,q}\,(\text{ax})}{p \vee \mathrm{G}\,q \vdash p, \mathrm{G}\,q}\,(\vee\vdash) \quad \dfrac{\dfrac{}{p \vdash p}\,(\text{ax}) \quad \dfrac{}{q \vdash q}\,(\text{ax})}{p \vee q \vdash p, q}\,(\vee\vdash) \quad \dfrac{\dfrac{}{q \vdash q}\,(\text{ax}) \quad \dfrac{}{\mathrm{G}\,p \vdash \mathrm{G}\,p}\,(\text{ax})}{q \vee \mathrm{G}\,p \vdash \mathrm{G}\,p, q}\,(\vee\vdash)}{\mathrm{G}\,(p \vee \mathrm{G}\,q), \mathrm{G}\,(p \vee q), \mathrm{G}\,(q \vee \mathrm{G}\,p) \vdash \mathrm{G}\,p, \mathrm{G}\,q}\,(\mathrm{G})$$
$$\vdots$$
$$\vdash \mathrm{G}\,(p \vee \mathrm{G}\,q) \wedge \mathrm{G}\,(p \vee q) \wedge \mathrm{G}\,(q \vee \mathrm{G}\,p) \supset \mathrm{G}\,p \vee \mathrm{G}\,q$$

FIGURE 2.4. Proof of the axiom .**3** in our calculus.

Finally, since our models are non-branching to the right and transitive, the rule must consider all the possible orders in which the formulæ φ_i will be realized in the future.

EXAMPLE 2.5. The axiom .**3** can be proved in our calculus, as shown in Figure 2.4. The first propositional steps are omitted, useless formulæ are discarded for better readability, and the admissible rule $(\vee\vdash)$ from Example 2.3 is used to deal will disjunctions on the left-hand side of a sequent.

Since the rule (G) can discard some formulæ, a natural proof search strategy is to only use it as a last resort: we only apply this rule if no other rule can be applied, and in such an instance we always take Γ' and Δ as small as possible, in order to not discard any modal formula. Following this strategy, our proof system still enjoys the invertibility property. Moreover, it also enjoys a finite branch property: the branches of a proof search can be polynomially bounded by the size of the input, since the premises of the new rule (G) only have subformulæ from its conclusion, and at least one formula of the form $\mathrm{G}\,\varphi$ on the right-hand side of the turnstile is unboxed. Finally, it has the failure property since a counter-model can be extracted from any failure branch: along such a branch, our strategy ensures that any time the rule (G) is applied, its conclusion is saturated for non modal formulæ. Thus, the counter-model we construct will have a world for every conclusion sequent of a (G) rule applied in the failure branch, with those worlds ordered as their corresponding sequents appear along the branch, and the valuation for every world can be constructed as for the propositional calculus.

Hence, this extended sequent calculus is sound and complete with respect to the logic **K4.3** [Lick, 2016], and with coNP proof search.

This illustrates how proof systems are modular: we obtained a new sequent calculus simply by adding a rule to the previous one, which was designed for a smaller logic.

2.4. Tense Logic on Linear Frames

We now focus on $\mathbf{K_t4.3}$ [Blackburn et al., 2001; Cocchiarella, 1965], the tense logic of linear frames, which also have a symmetrical H modality to express what holds in the past. Even though it is less expressive than LTL, according to Sistla and Clarke [1985], a large number of LTL formulæ fall into $\mathbf{K_t4.3}$ in practice. The former sequent calculus seems unfit to work with this extended logic, as we cannot just forget some information about the current world any more when jumping to the future: we might need this information later during the proof search when dealing with a past formula. Somewhat surprisingly for the logic lying at the heart of LTL with past modalities—which is largely studied in verification [Laroussinie et al., 2002; Lichtenstein et al., 1985]—, to the best of our knowledge, a sound and complete sequent-style calculus for $\mathbf{K_t4.3}$ was recently proposed by Indrzejczak [2016], for which cut elimination

has been proved by Indrzejczak [2017]. This is an *ordered hypersequent* calculus, where the structure of the hypersequents reflects the linear structure of $\mathbf{K_t4.3}$ frames. However, this calculus does not yield a proof-search algorithm, even though $\mathbf{K_t4.3}$ satisfiability is known to be decidable and even NP-complete [Ono and Nakamura, 1980, see Section 2.4.6].

In this section, we present Indrzejczak's calculus, and discuss what forbids us from obtaining an effective proof search algorithm. We start by recalling the definition of $\mathbf{K_t4.3}$.

2.4.1. Modal Logic on Weak Total Orders. We now consider tense logics with two unary temporal operators, over a set Φ of propositional variables, with the following syntax:

$$\varphi ::= \ \bot \mid p \mid \varphi \supset \varphi \mid \mathsf{G}\,\varphi \mid \mathsf{H}\,\varphi \qquad \text{(where } p \in \Phi\text{)}$$

A new type of modal formulæ of the form $\mathsf{H}\,\varphi$ is now available. Intuitively, just like $\mathsf{G}\,\varphi$ expresses that φ holds 'globally' in all future worlds reachable from the current one, $\mathsf{H}\,\varphi$ expresses that φ holds 'historically' in all past worlds from which the current world is accessible. Moreover, similarly to $\mathsf{F}\,\varphi = \neg\mathsf{G}\,\neg\varphi$ expressing that φ will hold 'in the future', we define $\mathsf{P}\,\varphi = \neg\mathsf{H}\,\neg\varphi$ expressing that φ was true 'in the past.'

2.4.2. Semantics. As in Section 2.3, our formulæ shall be evaluated on *Kripke structures*. A *frame* is a pair $\mathfrak{F} = (W, \precsim)$, where W is a set of worlds, and $\precsim \subseteq W \times W$ is a binary relation over W. A *structure* is a pair $\mathfrak{M} = (\mathfrak{F}, V)$, where $\mathfrak{F} = (W, \precsim)$ is a frame, and $V : \Phi \to 2^W$ is a valuation function. Given such a structure, we define the *satisfaction* relation $\mathfrak{M}, w \models \varphi$, where $w \in W$ and φ is a formula, by structural induction on φ:

$$\begin{array}{ll}
\mathfrak{M}, w \not\models \bot & \\
\mathfrak{M}, w \models p & \text{iff } w \in V(p) \\
\mathfrak{M}, w \models \varphi \supset \psi & \text{iff } \mathfrak{M}, w \models \varphi \text{ implies } \mathfrak{M}, w \models \psi \\
\mathfrak{M}, w \models \mathsf{G}\,\varphi & \text{iff } \forall w' \in W \text{ such that } w \precsim w', \mathfrak{M}, w' \models \varphi \\
\mathfrak{M}, w \models \mathsf{H}\,\varphi & \text{iff } \forall w' \in W \text{ such that } w' \precsim w, \mathfrak{M}, w' \models \varphi
\end{array}$$

When $\mathfrak{M}, w \models \varphi$, we say that $(\mathfrak{M}, w)$ is a *model* of φ.

A formula that is satisfied in all worlds of all structures is said to be *valid*. As in the previous section, we shall not consider the validity problem in general, but only over *weak total orders*, i.e., the formulæ that hold in all structures whose accessibility relation $\precsim$ is transitive and total. As for $\mathbf{K4.3}$, this logic can be defined axiomatically, as shown next. Later in Chapter 3, we will study further restrictions of the logic.

2.4.3. Axiomatisation. The logic $\mathbf{K_t4.3}$ is defined as the set of theorems generated by necessitation, modus ponens and substitution from classical tautologies and the axioms [Blackburn et al., 2001, p. 207]:

$$\mathsf{G}(p \supset q) \supset (\mathsf{G}\,p \supset \mathsf{G}\,q) \tag{$\mathbf{K_r}$}$$
$$\mathsf{H}(p \supset q) \supset (\mathsf{H}\,p \supset \mathsf{H}\,q) \tag{$\mathbf{K}_\ell$}$$
$$p \supset \mathsf{G}\,\mathsf{P}\,p \tag{$\mathbf{t_r}$}$$
$$p \supset \mathsf{H}\,\mathsf{F}\,p \tag{$\mathbf{t}_\ell$}$$
$$\mathsf{F}\,\mathsf{F}\,p \supset \mathsf{F}\,p \tag{$\mathbf{4}$}$$
$$\mathsf{F}\,p \wedge \mathsf{F}\,q \supset \mathsf{F}(p \wedge \mathsf{F}\,q) \vee \mathsf{F}(p \wedge q) \vee \mathsf{F}(q \wedge \mathsf{F}\,p) \tag{$.\mathbf{3_r}$}$$
$$\mathsf{P}\,p \wedge \mathsf{P}\,q \supset \mathsf{P}(p \wedge \mathsf{P}\,q) \vee \mathsf{P}(p \wedge q) \vee \mathsf{P}(q \wedge \mathsf{P}\,p) \tag{$.\mathbf{3}_\ell$}$$

The first two axioms are simply the Kripke schema, given for each modality. Next we find the (**t**) axioms, which are obviously satisfied in our setting since the two modalities are converses of each other.[1] Similarly to **K4.3**, (**4**) corresponds to the transitivity of $\precsim$; and the canonical models of the *trichotomy* axioms **.3** have accessibility relationships that are non-branching to the left and to the right. Altogether, this implies the following completeness result:

Fact 2.6 ([Blackburn et al., 2001, p. 222]). *A formula is a theorem of* $\mathbf{K_t4.3}$ *iff it is valid in all structures whose relation is transitive and total, i.e., in* weak total orders.

The logic $\mathbf{K_t4.3}$ is perhaps better known for being complete with respect to the class of *strict* total orders [Blackburn et al., 2001, Thm. 4.56]. As we will see later, the hypersequent calculus from Indrzejczak [2016] focuses on this characterisation.

2.4.4. First-Order Logic with Two Variables. The logic $\mathbf{K_t4.3}$ has been shown to be exactly as expressive as the two-variable fragment of first-order logic over linear orders by Manuel and Sreejith [2016]. We recall this result in this section.

2.4.4.1. *Syntax and Semantics.* We consider first-order formulæ with two variables x and y over the signature $(=, <, (p)_{p\in\Phi})$ where $=$ and $<$ are binary relational symbols and each p is a unary relational symbol:

$$\psi ::= z = z' \mid z < z' \mid p(z) \mid \bot \mid \psi \supset \psi \mid \forall z.\psi \qquad \text{(first-order formulæ)}$$

where z, z' range over $\{x, y\}$ and p over Φ. We call this logic $\mathrm{FO}^2(<)$.

We interpret our formulæ over structures $\mathfrak{M} = (W, <, V)$ where $=$ is interpreted as the equality over W, $<$ as the strict total ordering of W, and each p as $V(p)$ for the valuation $V : \Phi \to 2^W$.

[1] In a standard bi-modal setting, we would have two a priori unrelated relations. The **t** axioms would then force the two relations to be converses of each other in canonical models.

That is, we say that $\mathfrak{M}$ satisfies ψ under an assignment $\sigma : \{x, y\} \to W$, written $\mathfrak{M}, \sigma \models \psi$, in the following inductive cases:

$$
\begin{array}{ll}
\mathfrak{M}, \sigma \not\models \bot & \\
\mathfrak{M}, \sigma \models z = z' & \text{if } \sigma(z) = \sigma(z') \\
\mathfrak{M}, \sigma \models z < z' & \text{if } \sigma(z) < \sigma(z') \\
\mathfrak{M}, \sigma \models p(z) & \text{if } \sigma(z) \in V(p) \\
\mathfrak{M}, \sigma \models \psi \supset \psi' & \text{if } \mathfrak{M}, \sigma \models \psi \text{ implies } \mathfrak{M}, \sigma \models \psi' \\
\mathfrak{M}, \sigma \models \exists z.\psi & \text{if } \exists w \in W,\ \mathfrak{M}, \sigma[w/z] \models \psi
\end{array}
$$

where $\sigma[w/z]$ is the updated assignment mapping z to w and the remaining variable $z' \in \{x, y\} \setminus \{z\}$ to $\sigma(z')$.

2.4.4.2. *Equivalence with* $\mathbf{K_t4.3}$. Given an $\mathrm{FO}^2(<)$ formula $\varphi(z)$ with one free variable z, Etessami et al. [2002] show how to construct a $\mathbf{K_t4.3}$ formula φ' such that, for all strict totally ordered structures $\mathfrak{M} = (W, <, V)$, $\mathfrak{M}, [w/z] \models \varphi$ if and only if $\mathfrak{M}, w \models \varphi'$, where $[w/z]$ is the variable assignment mapping z to w.

Fact 2.7 ([Etessami et al., 2002, Thm. 2]). *Every* $\mathrm{FO}^2(<)$ *formula* $\varphi(x)$ *can be converted to an equivalent* $\mathbf{K_t4.3}$ *formula* φ' *with* $|\varphi'| \in 2^{poly(|\varphi|)}$.

Proof. We briefly recall how the proof works. The proof from Etessami et al. [2002] consist first in putting $\varphi(x)$ in Scott normal form, and then constructing its translation by structural induction. After multiples steps—involving an exponential blow-up—, they obtain the following formula equivalent to $\varphi(x)$:

$$\bigvee_{\bar{\gamma} \in \{\top, \bot\}^s} \left(\bigwedge_{i<s} (\xi_i(x) \leftrightarrow \gamma_i) \wedge \bigvee_{\tau \in \Upsilon} \exists y \,.\, (\tau(x, y) \wedge \beta^\tau(y, \bar{\gamma})) \right)$$

where each of the ξ_i have a *quantifier depth* strictly lower than φ (hence can be translated by induction hypothesis), and where $\tau(x, y)$ is what they call an *order type*, and expresses which relations hold between x and y. By β^τ, we denote the formula β where every atomic order formula have been replaced by either $\top$ or $\bot$, according to τ. At this point, assuming by induction hypothesis that ψ' is the translation of some formula $\psi(x)$, we need to provide a translation to a formula of the form $\exists y(\tau(x, y) \wedge \psi(y))$. They consider three mutually exclusive cases of such $\tau(x, y)$ in the following table, where $\tau\langle\psi\rangle$ denotes the translation of $\exists y(\tau(x, y) \wedge \psi(y))$:

$\tau(x, y)$	$\tau\langle\psi\rangle$
$x = y$	ψ'
$x < y$	$\mathsf{F}\,\psi'$
$y < x$	$\mathsf{P}\,\psi'$

□

Although the proof of Etessami et al. [2002, Thm. 2] is given for the case of the strict total order ω—i.e., for ω-words over the alphabet 2^Φ—, it actually does not rely on this specific frame and applies similarly to arbitrary strict total orders.

Hence, developing an optimal NP procedure for the satisfiability problem of $\mathbf{K_t4.3}$ will also lead to an optimal NEXP procedure for the satisfiability problem of $\mathrm{FO}^2(<)$. This will be developed in Section 3.4.

2.4.5. A First Proof System for $\mathbf{K_t4.3}$. Indrzejczak [2016] proposed a sound and complete calculus for $\mathbf{K_t4.3}$ using the framework of *ordered* hypersequents (aka. linear nested sequents [Lellmann, 2015]): his calculus works with *lists* of sequents rather than the usual *multisets* of sequents of hypersequent calculi. If $s_1, \cdots, s_n$ are sequents, we write $s_1 \,;\cdots;\, s_n$ to denote the hypersequent consisting of the ordered list of $s_1, \cdots, s_n$. The semantics of a sequent remains the same, and the semantics of ordered hypersequents relies on a mapping from ordered sequents to worlds that are ordered accordingly.

Definition 2.8 ([Indrzejczak, 2016, Def. 5.1]). For any $\mathbf{K_t4.3}$-model $\mathfrak{M}$ and hypersequent $H = s_1 \,;\cdots;\, s_n$, we say that $\mathfrak{M}$ is a model of H (written $\mathfrak{M} \models H$) iff for all worlds $t_1, \cdots, t_n$ of $\mathfrak{M}$, if $t_1 \precsim t_2 \precsim \cdots \precsim t_n$, then there exists $i \leq n$ such that $\mathfrak{M}, t_i \models s_i$.

Following that definition, a hypersequent H is valid if it is satisfied by every model. This extension allows for a natural calculus, enjoying the subformula property and extending nicely to accommodate semantic restrictions such as unboundedness and density.

The calculus rules from Indrzejczak [2016] are presented in Figures 2.5 and 2.6.

$$\frac{}{H_1 \,;\, \Gamma, \varphi \vdash \varphi, \Delta \,;\, H_2}\ (\mathsf{ax}) \qquad \frac{H_1 \,;\, \Gamma, \varphi \vdash \psi, \Delta \,;\, H_2}{H_1 \,;\, \Gamma \vdash \varphi \supset \psi, \Delta \,;\, H_2}\ (\vdash \supset)$$

$$\frac{H_1 \,;\, \Gamma \vdash \varphi, \Delta \,;\, H_2 \quad H_1 \,;\, \Gamma, \psi \vdash \Delta \,;\, H_2}{H_1 \,;\, \Gamma, \varphi \supset \psi \vdash \Delta \,;\, H_2}\ (\supset \vdash)$$

Figure 2.5. Propositional Rules from Indrzejczak [2016].

$$\frac{H \,;\, \Gamma \vdash \Delta; \vdash \varphi}{H \,;\, \Gamma \vdash \mathsf{G}\,\varphi, \Delta}\ (\vdash \mathsf{G}) \qquad \frac{H_1 \,;\, \Gamma \vdash \Delta \,;\cdots;\, \Pi, \varphi \vdash \Sigma \,;\, H_2}{H_1 \,;\, \Gamma, \mathsf{G}\,\varphi \vdash \Delta \,;\cdots;\, \Pi \vdash \Sigma \,;\, H_2}\ (\mathsf{G} \vdash)$$

$$\frac{\vdash \varphi \,;\, \Gamma \vdash \Delta \,;\, H}{\Gamma \vdash \mathsf{H}\,\varphi, \Delta \,;\, H}\ (\vdash \mathsf{H}) \qquad \frac{H_1 \,;\, \Pi, \varphi \vdash \Sigma \,;\cdots;\, \Gamma \vdash \Delta \,;\, H_2}{H_1 \,;\, \Pi \vdash \Sigma \,;\cdots;\, \Gamma, \mathsf{H}\,\varphi \vdash \Delta \,;\, H_2}\ (\mathsf{H} \vdash)$$

$$\frac{H_1 \,;\, \Gamma \vdash \Delta; \vdash \varphi \,;\, \Pi \vdash \Sigma \,;\, H_2 \quad H_1 \,;\, \Gamma \vdash \Delta \,;\, \Pi \vdash \varphi, \Sigma \,;\, H_2 \quad H_1 \,;\, \Gamma \vdash \Delta \,;\, \Pi \vdash \mathsf{G}\,\varphi, \Sigma \,;\, H_2}{H_1 \,;\, \Gamma \vdash \mathsf{G}\,\varphi, \Delta \,;\, \Pi \vdash \Sigma \,;\, H_2}\ (\vdash \mathsf{G}')$$

$$\frac{H_1 \,;\, \Pi \vdash \Sigma; \vdash \varphi \,;\, \Gamma \vdash \Delta \,;\, H_2 \quad H_1 \,;\, \Pi \vdash \varphi, \Sigma \,;\, \Gamma \vdash \Delta \,;\, H_2 \quad H_1 \,;\, \Pi \vdash \mathsf{H}\,\varphi, \Sigma \,;\, \Gamma \vdash \Delta \,;\, H_2}{H_1 \,;\, \Pi \vdash \Sigma \,;\, \Gamma \vdash \mathsf{H}\,\varphi, \Delta \,;\, H_2}\ (\vdash \mathsf{H}')$$

Figure 2.6. Modal Rules from Indrzejczak [2016].

For example, the calculus of Indrzejczak [2016] allows the following inference:

$$\frac{\Gamma \vdash \Delta; \ \vdash \varphi}{\Gamma \vdash \Delta, \mathsf{G}\,\varphi}\ (\vdash \mathsf{G})$$

It expresses that, if $w \not\Vdash \mathsf{G}\,\varphi$ for an arbitrary world w, there must be a $w \precsim w'$ such that $w' \not\Vdash \varphi$. In general, ($\vdash \mathsf{G}'$) may require several premises, but in this particular case just one premise with a new cell suffices.

A proof search in this calculus on an invalid hypersequent will, in a sense, try to build a bigger hypersequent representing a counter-model of the input. The hypersequents from Indrzejczak [2016] are well suited to represent strict total orders, but focusing on this characterisation is counterproductive for our purposes. As a simple illustration of when weak total orders could be beneficial, note that some formulæ admit finite weak total orders as models but only infinite strict total orders.

Example 2.9. It is the case, for example, of $(\mathsf{G}\,\mathsf{F}\top) \wedge (\mathsf{F}\top)$, which admits a single-world model that is a weak total order. The dual sequent, $\mathsf{G}\neg\mathsf{G}\bot \vdash \mathsf{G}\bot$, has finite counter-models with a weak total order, but no finite counter-models with a strict total order (a counter-model of this sequent must be unbounded to the right). When trying to prove this sequent with the calculus of Indrzejczak [2016], the proof search strategy underlying its completeness argument unfolds the following infinite derivation, by alternating the right and left introduction rules for G (with implicit uses of the left rules for $\neg$), as it is the only way to create an unbounded counter-model with a strict total order.

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\mathsf{G}\neg\mathsf{G}\bot \vdash \mathsf{G}\bot \;;\; \vdash \mathsf{G}\bot, \bot \;;\; \vdash \bot}}{\mathsf{G}\neg\mathsf{G}\bot \vdash \mathsf{G}\bot;\; \vdash \mathsf{G}\bot, \bot}}{\mathsf{G}\neg\mathsf{G}\bot \vdash \mathsf{G}\bot;\; \vdash \bot}}{\mathsf{G}\neg\mathsf{G}\bot \vdash \mathsf{G}\bot}$$

Principal formulas shown in orange, useless formulas in gray.

The calculus from Indrzejczak [2016] presented in this part enjoys cut elimination, as shown in Indrzejczak [2017]. This enables another type of proof of completeness: one only needs to prove the axioms of the logic in the proof system. However, such a proof of completeness does not provide any effective proof search algorithm, as opposed to the approach presented in Section 2.2.4. As a result, we will not consider cut elimination for the proof systems presented in this part, as our efforts are driven by algorithmic purposes.

2.4.6. Finite Model Property. The use of weak total orders is instrumental in order to derive decidability and complexity results, as it allows the logic to enjoy a *finite model property*. This result has been established by Ono and Nakamura [1980], not only for $\mathbf{K_t4.3}$ but also for all logics considered in Chapter 3. Finite models are obtained by using a filtration [Blackburn et al., 2001, Def. 2.36] on a structure to obtain a finite structure of the same 'shape.' The relevant filtration in this case is called the *Lemmon filtration*.

Definition 2.10 (Lemmon Filtration). Let $\mathfrak{M} = (W, \precsim, V)$ be a Kripke structure. Let Ψ be a set of $\mathbf{K_t4.3}$ formulæ closed under taking subformulæ. We define a binary relation $\equiv$ on W by:

$$w \equiv w' \text{ iff } \forall \psi \in \Psi,\ \mathfrak{M}, w \models \psi \iff \mathfrak{M}, w' \models \psi$$

The relation $\equiv$ is an equivalence relation, and we note $[w]$ the equivalence class of a world $w \in W$. Note that, if Ψ is finite, then $\equiv$ has finite index. Moreover, if $w \equiv w'$, then $\forall p \in \Phi \cap \Psi$, $w \in V(p) \iff w' \in V(p)$. Hence, we can define the *Lemmon filtration* of $\mathfrak{M}$ by Ψ as $\mathfrak{M}^\dagger = (W^\dagger, \precsim^\dagger, V^\dagger)$ such that:

$$W^\dagger = W/\equiv \qquad V^\dagger(p) = V(p)/\equiv$$

$$[w] \precsim^\dagger [w'] \text{ iff } \begin{cases} \forall \mathsf{G}\,\psi \in \Psi, \text{ if } \mathfrak{M}, w \models \mathsf{G}\,\psi \text{ then } \mathfrak{M}, w' \models \mathsf{G}\,\psi \text{ and } \mathfrak{M}, w' \models \psi \\ \forall \mathsf{H}\,\psi \in \Psi, \text{ if } \mathfrak{M}, w' \models \mathsf{H}\,\psi \text{ then } \mathfrak{M}, w \models \mathsf{H}\,\psi \text{ and } \mathfrak{M}, w \models \psi \end{cases}$$

FACT 2.11 ([Ono and Nakamura, 1980, Thm. 3]). *Let $\mathfrak{M} = (W, \precsim, V)$ be a weak total order and Ψ a set of $\mathbf{K_t4.3}$ formulæ closed under taking subformulæ, and let $\mathfrak{M}^\dagger = (W^\dagger, \precsim^\dagger, V^\dagger)$ be the Lemmon filtration of $\mathfrak{M}$ by Ψ. Then*

(i) $[w] \precsim^\dagger [w']$ if $w \precsim w'$,
(ii) $\precsim^\dagger$ is transitive and linear,
(iii) $\precsim^\dagger$ is unbounded to the right (resp. left) if $\precsim$ is unbounded to the right (resp. left), and
(iv) $\precsim^\dagger$ is dense if $\precsim$ is dense.

When Ψ is the finite set of subformulæ of a given formula φ, this filtration produces a finite model from any model of φ. Moreover, once a finite model is constructed for a satisfiable formula φ, we can also transform it into a *small* model of polynomial size in $|\varphi|$ by only keeping extremal positions realising modal subformulæ of φ. This last step establishes the NP-completeness of the satisfiability problem for $\mathbf{K_t4.3}$.

FACT 2.12 ([Blackburn et al., 2001, p. 379]). $\mathbf{K_t4.3}$ *has an* NP*-complete satisfiability problem.*

In the next chapter, we show how to modify Indrzejczak's proof system to make it leverage weak total orders, and thus take advantage of these model-theoretic results. In the hypersequent calculus we obtain, proof search branches can be polynomially bounded, which leads to an optimal coNP algorithm solving the validity problem of tense logic. In Chapter 4, we enrich our calculus to work with ordinals while still enjoying a coNP proof search. In Chapter 5, we enrich it further to work with data ordinals and freeze quantifiers. The logic we consider in this last chapter is undecidable, but we exhibit a fragment for which proof search is in coNP in our calculus, and which is expressively equivalent to $\mathrm{FO}^2(<, \sim)$ (see Section 5.6).

CHAPTER 3

Tense Logic over Linear Frames

Contents

3.1. Introduction

As discussed in Section 2.4.5, Indrzejczak [2016] proposed a complete calculus for $\mathbf{K_t4.3}$ using the framework of *ordered* hypersequents. Unfortunately, Indrzejczak's completeness argument is quite complex, and does not yield a decision procedure. The argument is Hintikka-style: if a careful exhaustive proof search fails in his calculus, then some failed proof-search branch yields a counter-model of the conclusion hypersequent. In Indrzejczak's calculus, that failure branch may be infinite, in which case the extracted counter-model is obtained as a limit, and is itself infinite. The issue here is that ordered hypersequents correspond to *strictly ordered* linear frames, which are arguably not the most adequate structures for the logic. Although every satisfiable $\mathbf{K_t4.3}$ formula has a model whose underling frame is a strict total order, there are examples of invalid formulæ (like the formula from Example 2.9), whose strictly ordered counter-models are all infinite. On such invalid instances, the hypersequent calculus of Indrzejczak [2016] yields a proof tree with some infinite failure branches, thus proof-search does not terminate. Instead, we should try to focus on weak total orders, which would be enough thanks to Fact 2.6.

The decidability of the satisfiability problem of $\mathbf{K_t4.3}$ comes from its finite model property, shown by Ono and Nakamura [1980, Thm. 3]. But this property can only be obtained when working with *weak* total orders, i.e. allowing some worlds of the models to be equivalent for the order relation. Such groups of nodes are commonly called 'clusters.' Note that the logic itself is not able to distinguish between a weakly ordered frame and any of its 'bulldozed' strict orders [Blackburn et al., 2001, Thm. 4.56].

In this chapter, we capture the syntactic aspects of these model-theoretic results. In Section 3.2, we show how to enhance the hypersequent calculus of Indrzejczak [2016] by capturing the model-theoretic ideas in hypersequents with *clusters* and *annotations*. This leads to a sound and complete proof system where proof search always terminates, furthermore with a coNP complexity—which is optimal for the validity problem. Moreover, this proof system is also modular: we consider some classical extensions of $\mathbf{K_t4.3}$ in Section 3.3, and provide new rules for our hypersequent calculus to handle these extensions; extended with these new rules, our proof system also yield an optimal coNP proof search for the corresponding extension of $\mathbf{K_t4.3}$. Finally, Manuel and Sreejith [2016] have recently shown that validity in first-order logic with two variables over strict total orders is in coNEXP. In Section 3.4, we derive the same statement from our results and extend it further to *dense* linear orders, by first converting the first-order formulæ into equivalent exponential-sized $\mathbf{K_t4.3}$ formulæ as recalled in Section 2.4.4 [Etessami et al., 2002].

3.2. Hypersequents with Clusters

3.2.1. Weak Total Orders. Our key insight is that capturing some aspects of weak total orders in our calculus could be beneficial. The choice of the symbol $\precsim$ for our frames' accessibility relations is in line with our focus on weak total orders. When working on such orders, it is useful to define $x \prec y$ when $x \precsim y$ but not $y \precsim x$. Note that $\prec$ may not be a strict total order: it is transitive but not necessarily total.

In that spirit, our proof system will manipulate another kind of formulæ called *annotations* in order to guide proof search. They consist of G or H formulæ written between parentheses and in violet, and their semantics relies on the relation $\prec$:

$$\mathfrak{M}, w \models (\mathsf{G}\,\varphi) \qquad \text{iff } \forall w' \in W \text{ such that } w \prec w', \mathfrak{M}, w' \models \varphi$$
$$\mathfrak{M}, w \models (\mathsf{H}\,\varphi) \qquad \text{iff } \forall w' \in W \text{ such that } w' \prec w, \mathfrak{M}, w' \models \varphi$$

It should be noted that the semantics of an annotation formula cannot be expressed otherwise in the original logic, since a model with clusters can be transformed to a bisimilar clusters-free model by 'bulldozing' its clusters, i.e. by replacing every cluster by an infinite strictly-ordered repetition of its worlds.

3.2.2. Finite Models and Hypersequents with Clusters. When a formula requires its strict total order models to be infinite, it is actually just forcing a finite number of configurations to be repeated ad infinitum. When working with weak total orders, such infinite sequences can be shrunk to a finite number of worlds—called a *cluster*—that are equivalent for the order relation, i.e. where $w \precsim w'$ and $w' \precsim w$ (noted $w \sim w'$) for all w, w' in the cluster. This is the reason why working with weak total orders allows our logic to enjoy a finite model property.

This insight leads us to consider ordered hypersequents with *syntactic clusters*, corresponding semantically to clusters inside a weak total order. This is where annotations become useful, as they can express what holds after the current cluster. At first glance, this seems to only complicate the calculus as it only creates more premises (and indeed some rules in our calculus have a large number of premises); but it will allow us to design a simple proof search strategy allowing us to bound failure branches. For example, as explained in Chapter 2, Indrzejczak [2016] allows the following inference:

$$\frac{\Gamma \vdash \Delta;\ \vdash \varphi}{\Gamma \vdash \Delta, \mathrm{G}\,\varphi}$$

This inference would be modified as follows:

$$\frac{\Gamma \vdash \Delta\ ;\ (\mathrm{G}\,\varphi) \vdash \varphi \quad \Gamma \vdash \Delta\ ;\ \{(\mathrm{G}\,\varphi) \vdash \varphi\}}{\Gamma \vdash \Delta, \mathrm{G}\,\varphi}$$

It still expresses that, if $w \not\Vdash \mathrm{G}\,\varphi$ for an arbitrary world w, there must be a $w \precsim w'$ such that $w' \not\Vdash \varphi$, but it now needs to consider two cases, corresponding to whether w' belongs to a cluster (right premise) or not (left premise). Moreover, we can assume that w' is a rightmost world such that $w' \not\Vdash \varphi$, meaning that we can assume that φ holds everywhere after the current cluster (if any). This exactly corresponds to assuming the annotation $(\mathrm{G}\,\varphi)$ holds at w'.

Viewing our hypersequent calculus as a search for counter-models, this corresponds to restricting this search for 'extremal' counter-models only. With this in place, we finally obtain a calculus where failure branches are finite. This allows for an elementary completeness argument, extracting finite weakly ordered counter-models from failure branches. This also allows to prove back the finite model property from Ono and Nakamura [1980, Thm. 4]. Another consequence is that proof search in our calculus directly yields an optimal coNP procedure for validity.

3.2.3. Definitions and Basic Meta-Theory. We shall now formally describe our calculus. We first define hypersequents with clusters and their semantics in terms of embeddings into weak total orders. We then present our system of deduction rules.

3.2.3.1. *Annotated Hypersequents with Clusters.* For the rest of this part, a *hypersequent* is a list of *cells*, each cell being either a sequent or a list of sequents called a (syntactic) *cluster*. We shall use the following abstract syntax, where both operators ';' and '‖' are associative with unit '•':

$$H ::= C \mid H\ ;\ H \qquad \text{(hypersequents)}$$

$$C ::= \bullet \mid S \mid \{Cl\} \qquad \text{(cells)}$$

$$Cl ::= S \mid Cl \parallel Cl \qquad \text{(cluster contents)}$$

Note that this definition allows for empty cells and hypersequents '•', but these notational conveniences will never arise in actual proofs—and should not be confused with the empty sequent '⊢'. The main feature of hypersequents with clusters is that their structures are weak total orders. The order of cells in a hypersequent *is* relevant, as it yields a strict ordering in the semantics. The order of sequents inside a cluster is semantically irrelevant; nevertheless, assuming an ordering as part of the syntactic structure of clusters is sometimes useful, as in the upcoming definition.

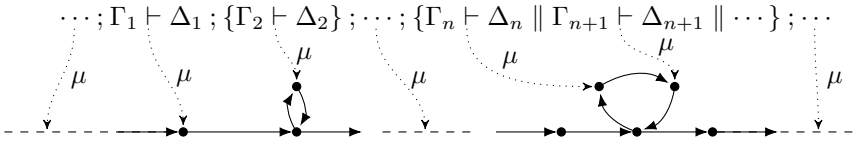

FIGURE 3.1. Embedding of a hypersequent in a weak total order.

3.2.3.2. *Underlying Frames and Embeddings.* We now introduce notations that allow to describe the relative positions between two sequents inside a hypersequent.

DEFINITION 3.1 (partial order of a hypersequent). Let H be a hypersequent containing n sequents, counting both the sequents found directly in its cells and those in its clusters. In this context, any $i \in [1; n]$ is called a *position* of H, and we write $H(i)$ for the i-th sequent of H. We define a partial order $\precsim$ on the positions of H by setting $i \precsim j$ if and only if either the i-th and j-th sequents are in the same cluster, or the i-th sequent is in a cell that lies strictly to the left of the cell of the j-th sequent. We write $i \prec j$ when $i \precsim j$ but $j \not\precsim i$, i.e. j lies strictly to the right of i in H. We write $i \sim j$ when $i \precsim j \precsim i$. Finally, the *domain* of H is defined as $\mathrm{dom}(H) = ([1; n], \precsim)$; note that empty cells are ignored in $\mathrm{dom}(H)$.

The domain of a hypersequent is actually a $\mathbf{K_t4.3}$-model. Thus, in this chapter, we may also call it the *underlying frame* of H.

DEFINITION 3.2. Let $\mathfrak{F} = (W, \precsim)$ and $\mathfrak{F}' = (W', \precsim')$ be two frames. We say that $\mu : W \to W'$ is an *embedding* of $\mathfrak{F}$ into $\mathfrak{F}'$ if, for all $(w_1, w_2) \in W^2$,

- $w_1 \precsim w_2$ implies $\mu(w_1) \precsim' \mu(w_2)$ and
- $\mu(w_1) \sim' \mu(w_2)$ implies $w_1 \sim w_2$.

In that case, we write $\mathfrak{F} \hookrightarrow_\mu \mathfrak{F}'$. We simply write $H \hookrightarrow_\mu \mathfrak{F}'$ when $\mathrm{dom}(H) \hookrightarrow_\mu \mathfrak{F}'$.

An example embedding is shown in Figure 3.1. Note that $\mu(i)$ cannot be reflexive when i is not. Likewise, positions from distinct cells cannot be embedded into worlds of a same cluster. By contrast, distinct positions belonging to the same cluster may be mapped to the same (reflexive) world.

DEFINITION 3.3 (semantics). Let $\mathfrak{M} = (\mathfrak{F}, V)$ be a structure. Given an embedding $H \hookrightarrow_\mu \mathfrak{F}$, we say that $(\mathfrak{M}, \mu)$ is a *model* of a hypersequent H, written $\mathfrak{M}, \mu \models H$, when there exists a position i of H such that $\mathfrak{M}, \mu(i) \models H(i)$. We say that a hypersequent is *valid* if for any weak total order $\mathfrak{M} = (\mathfrak{F}, V)$ and any embedding $H \hookrightarrow_\mu \mathfrak{F}$, we have $\mathfrak{M}, \mu \models H$.

Conversely, We say that $\mathfrak{M}$ is a *counter-model* of H if there exists an embedding μ such that $H \hookrightarrow_\mu \mathfrak{F}$ and for every position w of H, $\mathfrak{M}, \mu(w) \not\models H(w)$ holds. In that case, we write $\mathfrak{M} \not\models H$, or even $\mathfrak{M}, \mu \not\models H$ if we want to specify the embedding μ.

3.2.3.3. *Link with Validity of a Formula.* When testing for the validity of the hypersequent $\vdash \varphi$, we are a priori only testing whether φ holds in every $\mathbf{K_t4.3}$-model, in every position that is not in a cluster. However, when wanting to test for the validity of the formula φ, testing also for the validity of $\{\vdash \varphi\}$ is unnecessary, as the following result shows.

PROPOSITION 3.4. *If a formula φ has a counter-model $(\mathfrak{M}, w)$, then it has a counter-model $(\mathfrak{M}', w')$ such that $w' \not\sim w'$.*

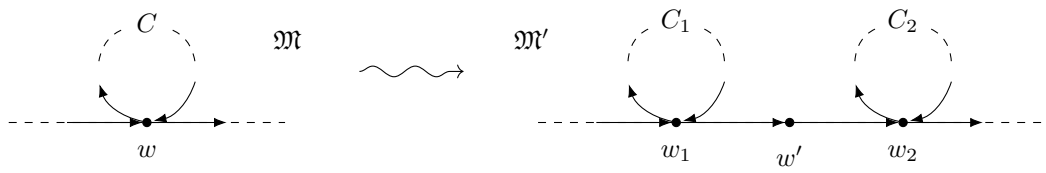

FIGURE 3.2. Duplication of the cluster in the proof of Proposition 3.4.

PROOF. If $w \not\sim w$, then we can take $\mathfrak{M}' = \mathfrak{M}$ and $w' = w$. Else, we can duplicate the cluster containing w, and putting a copy w' of w in between, as shown in Figure 3.2.

Formally, let $C = \{x \in W \mid x \sim w\}$ be the cluster containing w. We define a modified model $\mathfrak{M}' = (W', \precsim', V')$ featuring two copies of C and another copy of w as follows:

$$\begin{aligned}
W' &= (W \setminus C) \cup \{(x,b) \mid x \in C,\, b \in \{0,1\}\} \cup \{w'\} \\
V'(x) &= V(x)\ \forall x \in W \setminus C \\
V'(w') &= V(w) \\
V'((x,b)) &= V(x)\ \forall (x,b) \in C \times \{0,1\} \\
(x,b) &\precsim' (x',b)\ \forall (x,x',b) \in C^2 \times \{0,1\} \\
(x,0) &\precsim' (x',1)\ \forall (x,x') \in C^2 \\
(x,0) &\precsim' w'\ \forall x \in C \\
w' &\precsim' (x,1)\ \forall x \in C \\
x &\precsim' (x',b) \text{ whenever } x \precsim x' \\
(x,b) &\precsim' x' \text{ whenever } x \precsim x' \\
x &\precsim' x' \text{ whenever } x \precsim x'
\end{aligned}$$

We now have a $\mathbf{K_t4.3}$-model $\mathfrak{M}'$ with a cluster-free world w'. Moreover, it is easy to see that $(\mathfrak{M}, w)$ and $(\mathfrak{M}', w')$ are bisimilar [Blackburn et al., 2001, Thm. 2.20]. Hence, since $\mathfrak{M}, w \not\models \varphi$, we have $\mathfrak{M}', w' \not\models \varphi$. So $(\mathfrak{M}', w')$ is a counter-model of φ. □

Hence, a formula φ is valid if and only if the hypersequent $\vdash \varphi$ is valid.

3.2.4. Proof System. We now present our hypersequent calculus. The rules are given in Figures 3.3 to 3.5, making use of a few notations.

$$\frac{}{H\,[\varphi, \Gamma \vdash \Delta, \varphi]}\ (\mathsf{ax}) \qquad \frac{H\,[\varphi \supset \psi, \Gamma \vdash \Delta, \varphi] \quad H\,[\varphi \supset \psi, \psi, \Gamma \vdash \Delta]}{H\,[\varphi \supset \psi, \Gamma \vdash \Delta]}\ (\supset \vdash)$$

$$\frac{}{H\,[\Gamma, \bot \vdash \Delta]}\ (\bot) \qquad \frac{H\,[\varphi, \Gamma \vdash \Delta, \psi, \varphi \supset \psi]}{H\,[\Gamma \vdash \Delta, \varphi \supset \psi]}\ (\vdash \supset)$$

FIGURE 3.3. Propositional rules of the hypersequent calculus with clusters.

$$\frac{H\,[\mathsf{G}\,\varphi, \Gamma \vdash \Delta]\,[\varphi, \mathsf{G}\,\varphi, \Pi \vdash \Sigma]}{H\,[\mathsf{G}\,\varphi, \Gamma \vdash \Delta]\,[\Pi \vdash \Sigma]}\ (\mathsf{G}\vdash) \qquad \frac{H_1; \{Cl_\bullet \parallel \varphi, \mathsf{G}\,\varphi, \Gamma \vdash \Delta \parallel Cl'_\bullet\}; H_2}{H_1; \{Cl_\bullet \parallel \mathsf{G}\,\varphi, \Gamma \vdash \Delta \parallel Cl'_\bullet\}; H_2}\ (\{\mathsf{G}\vdash\})$$

$$\frac{H\,[\varphi, \mathsf{H}\,\varphi, \Pi \vdash \Sigma]\,[\mathsf{H}\,\varphi, \Gamma \vdash \Delta]}{H\,[\Pi \vdash \Sigma]\,[\mathsf{H}\,\varphi, \Gamma \vdash \Delta]}\ (\mathsf{H}\vdash) \qquad \frac{H_1; \{Cl_\bullet \parallel \varphi, \mathsf{H}\,\varphi, \Gamma \vdash \Delta \parallel Cl'_\bullet\}; H_2}{H_1; \{Cl_\bullet \parallel \mathsf{H}\,\varphi, \Gamma \vdash \Delta \parallel Cl'_\bullet\}; H_2}\ (\{\mathsf{H}\vdash\})$$

$$\frac{\begin{array}{ll} H_1\,;\,C\,[\Gamma \vdash \Delta, \mathsf{G}\,\varphi]\,;\,(\mathsf{G}\,\varphi) \vdash \varphi\,;\,C'\,;\,H_2 & \\ H_1\,;\,C\,[\Gamma \vdash \Delta, \mathsf{G}\,\varphi]\,;\,\{(\mathsf{G}\,\varphi) \vdash \varphi\}\,;\,C'\,;\,H_2 & \\ H_1\,;\,C\,[\Gamma \vdash \Delta, \mathsf{G}\,\varphi \parallel (\mathsf{G}\,\varphi) \vdash \varphi]\,;\,C'\,;\,H_2 & \text{if } C \neq \star \\ H_1\,;\,C\,[\Gamma \vdash \Delta, \mathsf{G}\,\varphi]\,;\,C' \ltimes (\vdash \mathsf{G}\,\varphi)\,;\,H_2 & \text{if } C' \neq \bullet \\ H_1\,;\,C\,[\Gamma \vdash \Delta, \mathsf{G}\,\varphi]\,;\,C' \ltimes ((\mathsf{G}\,\varphi) \vdash \varphi)\,;\,H_2 & \text{if } C' \neq \bullet \text{ and } C' \neq \{Cl\} \end{array}}{H_1\,;\,C\,[\Gamma \vdash \Delta, \mathsf{G}\,\varphi]\,;\,C'\,;\,H_2}\ (\vdash\mathsf{G})$$

$$\frac{\begin{array}{ll} H_2\,;\,C'\,;\,(\mathsf{H}\,\varphi) \vdash \varphi\,;\,C\,[\Gamma \vdash \Delta, \mathsf{H}\,\varphi]\,;\,H_1 & \\ H_2\,;\,C'\,;\,\{(\mathsf{H}\,\varphi) \vdash \varphi\}\,;\,C\,[\Gamma \vdash \Delta, \mathsf{H}\,\varphi]\,;\,H_1 & \\ H_2\,;\,C'\,;\,C\,[\Gamma \vdash \Delta, \mathsf{H}\,\varphi \parallel (\mathsf{H}\,\varphi) \vdash \varphi]\,;\,H_1 & \text{if } C \neq \star \\ H_2\,;\,C' \ltimes (\vdash \mathsf{H}\,\varphi)\,;\,C\,[\Gamma \vdash \Delta, \mathsf{H}\,\varphi]\,;\,H_1 & \text{if } C' \neq \bullet \\ H_2\,;\,C' \ltimes ((\mathsf{H}\,\varphi) \vdash \varphi)\,;\,C\,[\Gamma \vdash \Delta, \mathsf{H}\,\varphi]\,;\,H_1 & \text{if } C' \neq \bullet \text{ and } C' \neq \{Cl\} \end{array}}{H_2\,;\,C'\,;\,C\,[\Gamma \vdash \Delta, \mathsf{H}\,\varphi]\,;\,H_1}\ (\vdash\mathsf{H})$$

FIGURE 3.4. Modal rules of the hypersequent calculus with clusters.

First, we use hypersequents with *holes*. One-placeholder hypersequents, cells, and clusters are defined by the syntax:

$$H\,[] ::= H\,;\,C\,[]\,;\,H \qquad\qquad C\,[] ::= \star \mid \{\, Cl[] \,\}$$
$$Cl[] ::= Cl_\bullet \parallel \star \parallel Cl_\bullet \qquad\qquad Cl_\bullet ::= \bullet \mid Cl$$

Two-placeholder cells and hypersequents have two holes identified by $\star_1$ and $\star_2$:

$$H\,[]\,[] ::= H\,;\,C\,[]\,[]\,;\,H \mid H[\star_1]\,;\,H[\star_2]$$
$$C\,[]\,[] ::= \{\, Cl[\star_1] \parallel Cl[\star_2] \,\} \mid \{\, Cl[\star_2] \parallel Cl[\star_1] \,\}$$

As usual, $C\,[S]$ (resp. $C\,[Cl]$) denotes the same cell with S (resp. Cl) substituted for $\star$; two-placeholder cells and hypersequents with holes behave similarly. In terms of the frames underlying hypersequents with two holes, observe that the positions i and j associated resp. to $\star_1$ and $\star_2$ are such that $i \precsim j$.

In addition, we use a convenient notation for *enriching* a sequent: if S is a sequent $\Gamma \vdash \Delta$, then $S \ltimes (\Gamma' \vdash \Delta')$ is the sequent $\Gamma, \Gamma' \vdash \Delta, \Delta'$. Moreover, we sometimes need to enrich an arbitrary sequent of a cluster C with a sequent S; then $C \ltimes S$ denotes the cluster with its leftmost sequent enriched.

After the usual propositional rules of Figure 3.3, which operate locally on a sequent, we give in Figure 3.4 the introduction rules for modalities. The left introduction rules are symmetric for our two modalities. The first two, $(\mathsf{G}\vdash)$ and $(\{\mathsf{G}\vdash\})$, express that if $\mathsf{G}\,\varphi$ holds at some position, then $\mathsf{G}\,\varphi$ and φ must also hold at a position to its right in the underlying frame.

$$\frac{H_1\,[(\mathsf{G}\,\varphi), \Gamma \vdash \Delta]\,;\, H_2\,[\varphi, \mathsf{G}\,\varphi, \Pi \vdash \Sigma]}{H_1\,[(\mathsf{G}\,\varphi), \Gamma \vdash \Delta]\,;\, H_2\,[\Pi \vdash \Sigma]}\ ((\mathsf{G}))\qquad \frac{H_1\,[\varphi, \mathsf{H}\,\varphi, \Gamma \vdash \Delta]\,;\, H_2\,[(\mathsf{H}\,\varphi), \Pi \vdash \Sigma]}{H_1\,[\Gamma \vdash \Delta]\,;\, H_2\,[(\mathsf{H}\,\varphi), \Pi \vdash \Sigma]}\ ((\mathsf{H}))$$

FIGURE 3.5. Annotation rules of the hypersequent calculus with clusters.

Regarding the right introduction rules for modalities, let us start with the particular case where these modalities occur in extremal cells. In rule ($\vdash\mathsf{G}$), we introduce a formula $\mathsf{G}\,\varphi$ to the right of a principal sequent that is in the rightmost cell of the hypersequent. The premises cover all the ways in which a world could occur to the right of (the embedding of) the principal sequent:

- We always have to consider a possible new cell strictly further to the right.
- Alternatively, if $C \neq \star$, the active sequent belongs to a cluster and we need the last premise when φ is falsified in an arbitrary world of that cluster.

In any case, the subformula φ comes alongside the annotation $(\mathsf{G}\,\varphi)$ on the left-hand side of the sequent. Intuitively, this corresponds to assuming that the position at hand is a right-most position where we need to prove φ, as the annotation assumes that φ holds in every strict future position. Rule ($\vdash\mathsf{H}$) is, as expected, symmetric. The cases where the active sequent is not extremal follow the same idea but have extra premises corresponding to the case where φ is falsified in the next cell C' or beyond.

Finally, the annotations rules from Figure 3.5 are similar to the left modal rules, but do not allow to send the subformula in the same syntactic cluster as the active formula.

Note that our rules are formulated in an invertible style, keeping the principal formula in the premises. This eases the proof of completeness, where proof search induces a form of saturation. The following weakening rules are admissible in our system, and we shall use them implicitly in examples to avoid carrying around useless formulæ:

$$\frac{H\,[\Gamma \vdash \Delta]}{H\,[\Gamma, \varphi \vdash \Delta]}\ (\text{weak}\vdash)\qquad \frac{H\,[\Gamma \vdash \Delta]}{H\,[\Gamma \vdash \varphi, \Delta]}\ (\vdash\text{ weak})$$

We prove invertibility with respect to Definition 3.3.

LEMMA 3.5 (invertibility). *For any instance of a deduction rule where the conclusion hypersequent is valid, all premisses are also valid.*

PROOF SKETCH. Considering a rule instance with a counter-model $(\mathfrak{M}, \mu)$ of a premise H, we build a counter-model $(\mathfrak{M}, \mu')$ of the conclusion H'. Depending on the rule that is applied, H and H' will either have exactly the same structure, or H will have a new cell. Accordingly, we take μ' to be the restriction of μ to the positions of H' (and adapt it accordingly for the positions that have been shifted). It is indeed a proper embedding of H' into $\mathfrak{M}$. It is then easy to see that $(\mathfrak{M}, \mu')$ is a counter-model of H', since any sequent $H'(i)$ is contained in the corresponding sequent $H(j)$: $\mathfrak{M}, \mu(j) \not\models H(j)$ implies $\mathfrak{M}, \mu'(i) \not\models H'(i)$. □

EXAMPLE 3.6. We provide on the next page a proof of the hypersequent $\{\mathsf{H}\,p, \mathsf{G}\,p, p \vdash \mathsf{G}\,\mathsf{H}\,p\}$ in our system. At each inference, the principal formula is indicated in orange and weakenings are implicit.

$$\dfrac{\mathcal{P} \quad \mathcal{P}' \quad \dfrac{\dfrac{\dfrac{}{p, (\mathsf{H}\,p) \vdash p \,;\, \{\mathsf{H}\,p, \mathsf{G}\,p, p \vdash \|(\mathsf{G}\,\mathsf{H}\,p) \vdash\}}\,(\text{ax})}{(\mathsf{H}\,p) \vdash p \,;\, \{\mathsf{H}\,p, \mathsf{G}\,p, p \vdash \|(\mathsf{G}\,\mathsf{H}\,p) \vdash\}}\,(\mathsf{H}\vdash) \quad \dfrac{\dfrac{}{\{p, (\mathsf{H}\,p) \vdash p\} \,;\, \{\mathsf{H}\,p, \mathsf{G}\,p, p \vdash \|(\mathsf{G}\,\mathsf{H}\,p) \vdash\}}\,(\text{ax})}{\{(\mathsf{H}\,p) \vdash p\} \,;\, \{\mathsf{H}\,p, \mathsf{G}\,p, p \vdash \|(\mathsf{G}\,\mathsf{H}\,p) \vdash\}}\,(\mathsf{H}\vdash) \quad \dfrac{\dfrac{}{\{\mathsf{H}\,p, \mathsf{G}\,p, p \vdash \|(\mathsf{G}\,\mathsf{H}\,p) \vdash\| \; p, (\mathsf{H}\,p) \vdash p\}}\,(\text{ax})}{\{\mathsf{H}\,p, \mathsf{G}\,p, p \vdash \|(\mathsf{G}\,\mathsf{H}\,p) \vdash\| \; (\mathsf{H}\,p) \vdash p\}}\,(\mathsf{G}\vdash)}{\{\mathsf{H}\,p, \mathsf{G}\,p, p \vdash \|(\mathsf{G}\,\mathsf{H}\,p) \vdash \mathsf{H}\,p\}}\,(\vdash\mathsf{H})}{\{\mathsf{H}\,p, \mathsf{G}\,p, p \vdash \mathsf{G}\,\mathsf{H}\,p\}}\,(\vdash\mathsf{G})$$

where $\mathcal{P}$ is:

$$\dfrac{\dfrac{\dfrac{}{\{\mathsf{H}\,p, \mathsf{G}\,p, p \vdash\} \,;\, p, (\mathsf{H}\,p) \vdash p \,;\, (\mathsf{G}\,\mathsf{H}\,p) \vdash}\,(\text{ax})}{\{\mathsf{H}\,p, \mathsf{G}\,p, p \vdash\} \,;\, (\mathsf{H}\,p) \vdash p \,;\, (\mathsf{G}\,\mathsf{H}\,p) \vdash}\,(\mathsf{G}\vdash) \quad \dfrac{\dfrac{}{\{\mathsf{H}\,p, \mathsf{G}\,p, p \vdash\} \,;\, \{p, (\mathsf{H}\,p) \vdash p\} \,;\, (\mathsf{G}\,\mathsf{H}\,p) \vdash}\,(\text{ax})}{\{\mathsf{H}\,p, \mathsf{G}\,p, p \vdash\} \,;\, \{(\mathsf{H}\,p) \vdash p\} \,;\, (\mathsf{G}\,\mathsf{H}\,p) \vdash}\,(\mathsf{G}\vdash) \quad \dfrac{}{\{\mathsf{H}\,p, \mathsf{G}\,p, p \vdash \mathsf{H}\,p\} \,;\, (\mathsf{G}\,\mathsf{H}\,p) \vdash}\,(\text{ax})}{\{\mathsf{H}\,p, \mathsf{G}\,p, p \vdash\} \,;\, (\mathsf{G}\,\mathsf{H}\,p) \vdash \mathsf{H}\,p}\,(\vdash\mathsf{H})$$

and where $\mathcal{P}'$ is:

$$\dfrac{\dfrac{\dfrac{}{\{\mathsf{H}\,p, \mathsf{G}\,p, p \vdash\} \,;\, p, (\mathsf{H}\,p) \vdash p \,;\, \{(\mathsf{G}\,\mathsf{H}\,p) \vdash\}}\,(\text{ax})}{\{\mathsf{H}\,p, \mathsf{G}\,p, p \vdash\} \,;\, (\mathsf{H}\,p) \vdash p \,;\, \{(\mathsf{G}\,\mathsf{H}\,p) \vdash\}}\,(\mathsf{G}\vdash) \quad \dfrac{\dfrac{}{\{\mathsf{H}\,p, \mathsf{G}\,p, p \vdash\} \,;\, \{p, (\mathsf{H}\,p) \vdash p\} \,;\, \{(\mathsf{G}\,\mathsf{H}\,p) \vdash\}}\,(\text{ax})}{\{\mathsf{H}\,p, \mathsf{G}\,p, p \vdash\} \,;\, \{(\mathsf{H}\,p) \vdash p\} \,;\, \{(\mathsf{G}\,\mathsf{H}\,p) \vdash\}}\,(\mathsf{G}\vdash) \quad \dfrac{\dfrac{}{\{\mathsf{H}\,p, \mathsf{G}\,p, p \vdash\} \,;\, \{(\mathsf{G}\,\mathsf{H}\,p) \vdash\| \; p, (\mathsf{H}\,p) \vdash p\}}\,(\text{ax})}{\{\mathsf{H}\,p, \mathsf{G}\,p, p \vdash\} \,;\, \{(\mathsf{G}\,\mathsf{H}\,p) \vdash\| \; (\mathsf{H}\,p) \vdash p\}}\,(\mathsf{G}\vdash) \quad \dfrac{}{\{\mathsf{H}\,p, \mathsf{G}\,p, p \vdash \mathsf{H}\,p\} \,;\, \{(\mathsf{G}\,\mathsf{H}\,p) \vdash\}}\,(\text{ax})}{\{\mathsf{H}\,p, \mathsf{G}\,p, p \vdash\} \,;\, \{(\mathsf{G}\,\mathsf{H}\,p) \vdash \mathsf{H}\,p\}}\,(\vdash\mathsf{H})$$

3.2.5. Soundness. We now show the soundness of our calculus.

LEMMA 3.7 (soundness). *All the rules of our hypersequent calculus with clusters are sound: if the premises of a rule instance are valid, then so is its conclusion.*

PROOF. We prove the contrapositive. Considering a rule instance whose conclusion H admits a counter-model $(\mathfrak{M}, \mu)$, we show that one of its premises also admits a counter-model $(\mathfrak{M}, \mu')$. Because of Fact 2.11, we can assume that $\mathfrak{M}$ is finite.

The cases of propositional rules are simple, and we focus on the other rules.

We first consider the case of rule ($\vdash$G), applied on a principal sequent $\Gamma \vdash \Delta, \mathsf{G}\,\varphi$ at position i in H. Since $\mathfrak{M}, \mu(i) \not\models \mathsf{G}\,\varphi$, there exists w' such that $\mu(i) \precsim w'$ and $\mathfrak{M}, w' \models \neg\varphi$. Since $\mathfrak{M}$ is finite we can take w' to be a rightmost world invalidating φ, i.e., such that there is no $w' \prec w''$ such that $w'' \models \neg\varphi$.

- We first consider the case where $\mu(i)$ and w' are two worlds (distinct or not) of the same cluster. Because of the last condition of Definition 3.2, i must be in a cluster in the underlying frame of H, so $C \neq \star$ and the premise $H_1 ; C\,[\Gamma \vdash \Delta, \mathsf{G}\,\varphi \parallel (\mathsf{G}\,\varphi) \vdash \varphi] ; C' ; H_2$ is available. We extend μ into μ', mapping the new sequent, at position $i+1$, to the world w': $\mu'(k) = \mu(k)$ for all $k \leq i$, $\mu'(i+1) = w'$, and $\mu'(k+1) = \mu(k)$ for all $k > i$. Then $(\mathfrak{M}, \mu')$ is a counter-model of the premise. In particular, the annotation $(\mathsf{G}\,\varphi)$ at position $i+1$ is respected, as we have chosen $\mu'(i+1) = w'$ such that for any $\mu'(i+1) \prec w''$, $w'' \models \varphi$.
- Otherwise, $\mu(i) \prec w'$. Let j be the first position in the cell C'. If $w' \prec \mu(j)$, we obtain a counter-model of either the premise $H_1 ; C\,[\Gamma \vdash \Delta, \mathsf{G}\,\varphi] ; (\mathsf{G}\,\varphi) \vdash \varphi ; C' ; H_2$ or the premise $H_1 ; C\,[\Gamma \vdash \Delta, \mathsf{G}\,\varphi] ; \{(\mathsf{G}\,\varphi) \vdash \varphi\} ; C' ; H_2$ (depending on whether w' is in a cluster) by adapting μ into an embedding μ' that assigns w' to the new position. If $\mu(j) \precsim w'$ then we have a counter-model of the fourth premise $H_1 ; C\,[\Gamma \vdash \Delta, \mathsf{G}\,\varphi] ; C' \ltimes (\vdash \mathsf{G}\,\varphi) ; H_2$, with the same embedding μ. Otherwise, $\mu(j) = w'$ and $\mu(j)$ is not reflexive, hence the last premise is available, namely $H_1 ; C\,[\Gamma \vdash \Delta, \mathsf{G}\,\varphi] ; C' \ltimes ((\mathsf{G}\,\varphi) \vdash \varphi) ; H_2$. Our counter-model $(\mathfrak{M}, \mu)$ is a counter-model of that premise.

We now consider the case of rule (G$\vdash$) applied on two positions $i \prec j$ in H. Since $(\mathfrak{M}, \mu)$ is a counter-model of H, then $\mathsf{G}\,\varphi$ holds at $\mu(i)$, so φ and $\mathsf{G}\,\varphi$ both hold at $\mu(j)$. The cases of ($\{\mathsf{G}\vdash\}$) and ((G)) are similar.

Finally, the cases of the past rules are analogous. □

3.2.6. Completeness and Complexity. We now turn to establishing completeness for our calculus, and to showing that proof search yields an optimal coNP procedure for deciding $\mathbf{K_t4.3}$ validity. These results follow from two properties of our calculus: deduction rules are invertible wrt. the semantics (recall Lemma 3.5), and proof search branches are polynomially bounded (as shown next in Lemma 3.11).

We start by identifying a shape of hypersequents that can always be proved.

LEMMA 3.8. *If a hypersequent H satisfies one of these conditions, then H is provable.*

(a) There exists a formula φ, and two positions $i \prec j$ of H such that $H(i)$ and $H(j)$ both contain the sequent $(\mathsf{G}\,\varphi) \vdash \varphi$.

(b) There exists a formula φ, and two positions $i \prec j$ of H such that $H(i)$ and $H(j)$ both contain the sequent $(\mathsf{H}\,\varphi) \vdash \varphi$.

In such a case, we say that H is immediately provable.

PROOF. For each case, we show how to prove H.

(a) Such a hypersequent can be proved as follows:

$$\dfrac{\dfrac{}{H_1\,[\Gamma, (\mathsf{G}\,\varphi) \vdash \varphi, \Delta]\,;\, H_2\,[\Gamma', (\mathsf{G}\,\varphi), \mathsf{G}\,\varphi, \varphi \vdash \varphi, \Delta']}\,(\text{ax})}{H_1\,[\Gamma, (\mathsf{G}\,\varphi) \vdash \varphi, \Delta]\,;\, H_2\,[\Gamma', (\mathsf{G}\,\varphi) \vdash \varphi, \Delta']}\,((\mathsf{G}))$$

(b) This case is similar, roles of i and j being reverted, and using $((\mathsf{H}))$ instead of $((\mathsf{G}))$:

$$\dfrac{\dfrac{}{H_1\,[\Gamma, (\mathsf{H}\,\varphi), \varphi \vdash \varphi, \Delta]\,;\, H_2\,[\Gamma', (\mathsf{H}\,\varphi) \vdash \varphi, \Delta']}\,(\text{ax})}{H_1\,[\Gamma, (\mathsf{H}\,\varphi) \vdash \varphi, \Delta]\,;\, H_2\,[\Gamma', (\mathsf{H}\,\varphi) \vdash \varphi, \Delta']}\,((\mathsf{H}))$$

□

During proof search, it is usual to require that the conclusion hypersequent of any rule application differs from all of the premisses of that rule. This amounts to forbidding useless proof search steps: no information would be gained from applying such a rule. For our calculus, we forbid another case, as no information is gained when a new sequent is created inside a cluster already containing another sequent with the same information.

DEFINITION 3.9. We call *partial proof* a finite derivation tree whose internal nodes correspond to rule applications, but whose leaves may be unjustified hypersequents, and that satisfies two conditions:

(a) no rule application should be such that, if H is the conclusion hypersequent,
   (i) one of the premises is also H, or
   (ii) the rule being applied is ($\vdash\mathsf{G}$) on a formula $\mathsf{G}\,\varphi$ at position i such that there exists $j \sim i$ such that $H(j)$ contains $(\mathsf{G}\,\varphi) \vdash \varphi$, or
   (iii) the rule being applied is ($\vdash\mathsf{H}$) on a formula $\mathsf{H}\,\varphi$ at position i such that there exists $j \sim i$ such that $H(j)$ contains $(\mathsf{H}\,\varphi) \vdash \varphi$.

(b) immediately provable hypersequents must be proven immediately as sketched in the proof of Lemma 3.8.

Finally, we call *failure hypersequent* a hypersequent on which any rule application would not respect condition (a).

PROPOSITION 3.10. *Any failure hypersequent has a counter-model.*

PROOF. Let H be a failure hypersequent, and let $\mathfrak{F} = (W, \precsim)$ be the underlying frame of H. Let $V : \Phi \to 2^W$ be the valuation defined for all $p \in \Phi$ by

$$V(p) = \{i \in W \mid p \text{ appears on the left-hand side of } H(i)\}\,.$$

Finally, let $\mathfrak{M} = (\mathfrak{F}, V)$. We shall establish that $(\mathfrak{M}, \mu)$ is a counter-model of H, where μ is the identity embedding $H \hookrightarrow_\mu \mathfrak{F}$. More precisely, we prove by structural induction on φ that, for every position i of H:

- If φ appears on the left of the turnstile in $H(i)$, then $\mathfrak{M}, i \models \varphi$.
- If φ appears on the right of the turnstile in $H(i)$, then $\mathfrak{M}, i \not\models \varphi$.

We reason by case analysis on φ.

- Case $\varphi = \bot$: $\bot$ never appears on the left-hand side of a sequent of H since the rule ($\bot$) cannot be applied, and $\mathfrak{M}, i \not\models \bot$ always holds.
- Case $\varphi = p \in \Phi$: immediate by definition of V and since the rule (ax) cannot be applied on H.

- Case $\varphi = \varphi_1 \supset \varphi_2$:
  - If φ appears on the left-hand side of $H(i)$ for some i, then, since the rule ($\supset \vdash$) cannot be applied on H, either φ_2 appears on the left-hand side of $H(i)$, or φ_1 appears on the right-hand side of $H(i)$. So, by induction hypothesis, either $\mathfrak{M}, i \models \varphi_2$ or $\mathfrak{M}, i \not\models \varphi_1$. Either way, $M, i \models \varphi_1 \supset \varphi_2$.
  - If φ appears on the right-hand side of $H(i)$ for some i, then, since the rule ($\vdash \supset$) cannot be applied on H, $H(i)$ contains the sequent $\varphi_1 \vdash \varphi_2$. So, by induction hypothesis, $\mathfrak{M}, i \models \varphi_1$ and $\mathfrak{M}, i \not\models \varphi_2$, so $\mathfrak{M}, i \not\models \varphi_1 \supset \varphi_2$.
- Case $\varphi = \mathsf{G}\,\varphi'$:
  - If $\mathsf{G}\,\varphi'$ appears on the left-hand side of a sequent $H(i)$, then, since rules ($\mathsf{G}\vdash$) and ($\{\mathsf{G}\vdash\}$) cannot be applied on H, φ' appears on the left-hand side of every sequent $H(j)$ such that $i \precsim j$. By induction hypothesis, $\mathfrak{M}, j \models \varphi'$ for all $i \precsim j$. Hence $\mathfrak{M}, i \models \mathsf{G}\,\varphi'$.
  - We now consider the case where $\mathsf{G}\,\varphi'$ appears on the right-hand side of $H(i)$. Let us first assume that there does not exist any $i \prec j$ such that $\mathsf{G}\,\varphi'$ appears on the right-hand side of $H(j)$.
    - If ($\vdash\mathsf{G}$) does not apply on φ because of case (i) of condition (a), it cannot be because of the fourth premise by the previous assumption, so an annotation $(\mathsf{G}\,\varphi')$ must appear at some position j in H on the left-hand side of the turnstile, along with φ' on its right-hand side, with $i \precsim j$. By induction hypothesis, $\mathfrak{M}, j \not\models \varphi'$, thus $\mathfrak{M}, i \not\models \mathsf{G}\,\varphi'$.
    - If ($\vdash\mathsf{G}$) does not apply on ψ because of case (ii) of condition (a), there exists $j \sim i$ such that $H(j)$ contains $(\mathsf{G}\,\varphi) \vdash \varphi$, and we can conclude in the same way we did above for case (i).

    Now, if there exists a position $i \prec j$ such that $\mathsf{G}\,\varphi'$ appears on the right-hand side of $H(j)$, we can choose j to be a right-most such position. And since we just proved that $\mathfrak{M}, j \not\models \mathsf{G}\,\varphi'$, then we indeed have $\mathfrak{M}, i \not\models \mathsf{G}\,\varphi'$.
- The case $\varphi = \mathsf{H}\,\varphi'$ is symmetric.
- Annotations $(\mathsf{G}\,\varphi')$ or $(\mathsf{H}\,\varphi')$ can only appear to the left of the turnstile, and their cases are analogous to the ones of $\mathsf{G}\,\varphi'$ or $\mathsf{H}\,\varphi'$. □

In general, proof search may diverge by expanding partial proofs infinitely, or require backtracking due to (finite) choices in rule applications.

Lemma 3.5 shows that backtracking is not necessary. We now turn to establishing that proof search terminates, and always produces branches of polynomial length. For a hypersequent H, let $\mathsf{len}(H)$ be its number of sequents (i.e., the size of $\mathrm{dom}(H)$), and $|H|$ the number of distinct subformulæ occurring in H.

Lemma 3.11 (small branch property). *For any partial proof of a hypersequent H, any branch of the proof is of length at most $2(|H| + \mathsf{len}(H) + 1) \cdot |H|$.*

Proof. Let H be a hypersequent, $\mathcal{P}$ a partial proof of it, and B a branch of $\mathcal{P}$. Remark that the number of positions in hypersequents of β is bounded by $|H|+\mathsf{len}(H)+1$: we have at most $\mathsf{len}(H)$ positions initially, and a new position may only be created once per modal formula among at most $|H|$ formulæ plus possibly one more (overall) to create an immediately provable hypersequent. This is because a second cell created by ($\vdash\mathsf{H}$) on the same $\mathsf{H}\,\varphi$ or by ($\vdash\mathsf{G}$) on the same $\mathsf{G}\,\varphi$ would belong to an immediately provable sequent, and because conditions (ii)

and (iii) prevent the same annotation from appearing twice inside the same cluster. Any rule application adds some subformulæ among $|H|$ to the left or to the right of the turnstile at a position among $|H| + \mathsf{len}(H) + 1$, hence with $2(|H| + \mathsf{len}(H) + 1) \cdot |H|$ choices. Thus B is of length at most $2(|H| + \mathsf{len}(H) + 1) \cdot |H|$. □

Lemma 3.11 shows that divergence cannot happen with our calculus, regardless of the way rules are applied. Hence, proof search in our calculus simply consists in expanding one proof attempt, either reaching a complete proof or obtaining a partial proof with at least one open leaf that cannot be derived by any rule application. Moreover, along with Proposition 3.10 we can recover the small model property from Blackburn et al. [2001, p. 379]: any satisfiable formula has a model of polynomial size.

Example 3.12. For instance, Example 2.9 considered the hypersequent $\mathsf{G}\neg\mathsf{G}\,\bot \vdash \mathsf{G}\,\bot$, which has finite counter-models with a weak total order, but no finite counter-models with a strict total order (a counter-model of this sequent must be unbounded to the right). When trying to prove this sequent with the calculus of Indrzejczak [2016], the proof search was unfolding an infinite derivation.

In our calculus, a derivation of that same hypersequent would necessarily contain several branches. The analogue of the one shown above can be closed thanks to the annotations:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{G}\neg\mathsf{G}\,\bot \vdash \mathsf{G}\,\bot ; (\mathsf{G}\,\bot) \vdash \mathsf{G}\,\bot, \bot ; (\mathsf{G}\,\bot), \bot \vdash \bot}\,(\mathrm{ax})}{\mathsf{G}\neg\mathsf{G}\,\bot \vdash \mathsf{G}\,\bot ; (\mathsf{G}\,\bot) \vdash \mathsf{G}\,\bot, \bot ; (\mathsf{G}\,\bot) \vdash \bot}\,(\vdash\mathsf{G})}{\mathsf{G}\neg\mathsf{G}\,\bot \vdash \mathsf{G}\,\bot ; (\mathsf{G}\,\bot) \vdash \mathsf{G}\,\bot, \bot}\,(\mathsf{G}\vdash) \qquad \cdots}{\mathsf{G}\neg\mathsf{G}\,\bot \vdash \mathsf{G}\,\bot ; (\mathsf{G}\,\bot) \vdash \bot}\,(\vdash\mathsf{G}) \qquad \cdots}{\mathsf{G}\neg\mathsf{G}\,\bot \vdash \mathsf{G}\,\bot}\,(\vdash\mathsf{G})$$

However, the following branch will lead to a failure hypersequent:

$$\dfrac{\dfrac{\mathsf{G}\neg\mathsf{G}\,\bot \vdash \mathsf{G}\,\bot ; \{(\mathsf{G}\,\bot) \vdash \mathsf{G}\,\bot, \bot\}}{\mathsf{G}\neg\mathsf{G}\,\bot \vdash \mathsf{G}\,\bot ; \{(\mathsf{G}\,\bot) \vdash \bot\}}\,(\mathsf{G}\vdash) \qquad \cdots}{\mathsf{G}\neg\mathsf{G}\,\bot \vdash \mathsf{G}\,\bot}\,(\vdash\mathsf{G})$$

On this last hypersequent, applying $(\vdash\mathsf{G})$ on the orange formula $\mathsf{G}\,\bot$ would create the premise

$$\mathsf{G}\neg\mathsf{G}\,\bot \vdash \mathsf{G}\,\bot ; \{(\mathsf{G}\,\bot) \vdash \mathsf{G}\,\bot, \bot \parallel (\mathsf{G}\,\bot) \vdash \bot\}$$

which is not allowed by our proof strategy: any other rule would have the same hypersequent as a premise.

In other words, it is a finite failure branch. As shown in Proposition 3.10, we can extract from it a finite counter-model featuring a reflexive world.

We can finally establish our completeness result.

Theorem 3.13 (completeness). *Our hypersequent calculus with clusters is complete: every valid hypersequent H has a proof.*

Proof. Assume that a hypersequent H is not provable. Consider a partial proof $\mathcal{P}$ of H that cannot be expanded any more. Such a partial proof is finite by Lemma 3.11; and at least one of its leaves is a failure hypersequent (or else $\mathcal{P}$ would be a proof of H). Moreover, this failure hypersequent has a counter-model by Proposition 3.10. Finally, by Lemma 3.5, the rules of our calculus are invertible so H also has a counter-model. □

We conclude by showing that proof search has an optimal complexity.

Proposition 3.14. *Proof search in our hypersequent calculus is in* coNP.

Proof. Proof search can be implemented in an alternating Turing machine maintaining the current hypersequent on its tape, where existential states choose which rule to apply to which principal sequent(s) and formula, and universal states choose a premise of the rule. By Lemma 3.11, the computation branches are of length bounded by a polynomial. By Lemma 3.5, the non-deterministic choices in existential states can be replaced by arbitrary deterministic choices, thus this Turing machine has only universal states, hence is in coNP. □

3.3. Extensions

The logic $\mathbf{K_t4.3}$ can be extended by additional axioms to further restrict the class of frames. We consider here two examples of such extensions also considered by Indrzejczak [2016]: density and unboundedness. For each extension, we show that our calculus can be adapted by adding new rules corresponding to the new axioms, and yields the same coNP upper bound. These new rules are rather different from Indrzejczak's, and exploit our use of hypersequents with clusters. Together, these rules extend our calculus into a sound and complete proof system with a coNP proof search algorithm for $\mathbf{K_tQ}$, the logic of *dense unbounded* linear frames, consisting of $\mathbf{K_t4.3}$ with both extensions.

Density. A frame $\mathfrak{F} = (W, \precsim)$ is *dense* if $\forall (x, y) \in W^2$, if $x \precsim y$ then $\exists z \in W$ such that $x \precsim z \precsim y$. Density is axiomatised by adding the following axiom:

$$\mathsf{F}\, p \supset \mathsf{F}\,\mathsf{F}\, p \qquad \textbf{(Den)}$$

This new logic also has a finite model property as well as a small model property [Ono and Nakamura, 1980]. Moreover, a finite weak total order is dense if and only if it never has two consecutive worlds that are not in clusters. This last property leads to the following new rule for our calculus to handle density:

$$\frac{H\,[S_1\,;\,\{\vdash\}\,;\,S_2]}{H\,[S_1\,;\,S_2]}\ (\mathsf{den})$$

Proposition 3.15. *Adding* (den) *to our calculus yields a sound and complete proof system for* $\mathbf{K_t4.3} \cup$ **(Den)**, *where proof search is in* coNP.

Proof. Our rule is obviously sound, as it closely reflects the shape of dense finite weak total orders: if the conclusion of this rule has a dense counter-model, the embedding must map S_1 and S_2 to two positions that are not in a cluster, hence there must be a cluster in between. It is also invertible, since the underlying frame of the conclusion of the rule is always a subframe of its premises. Hence Lemma 3.7 and Lemma 3.5 still hold.

To obtain that proof search is in coNP, it suffices to check that Lemma 3.11 carries over to our extension. This is true because the rule (den) can only be applied on two consecutive non-cluster cells, and whenever the rule (den) is applied on such a bad occurrence, this occurrence is no longer present in the premises. Hence, every time the rule (den) is applied, we reduce at least by one the number of bad occurrences, so we can only apply the rule (den) a finite number of times between applications of other rules creating new cells such as ($\vdash$G) and ($\vdash$H). Finally, since new cells can only be created polynomialy many times by those other rules thanks to our

initial strategy, the new rule (den) can, in the end, only be applied polynomialy many times along a branch. So the branches of our proof tree are still polynomial.

Finally, completeness is obtained as in Theorem 3.13. It only remains to show that the underlying frame of a failure hypersequent is dense. Indeed, if its underlying frame was not dense, we could apply the rule (den) which would contradict the fact that no rules can be applied any more on this hypersequent. □

Unboundedness. A frame $\mathfrak{F} = (W, \preceq)$ is *unbounded to the right* if $\forall x \in W, \exists y \in W$ such that $x \precsim y$. Symmetrically, a frame $\mathfrak{F} = (W, \precsim)$ is *unbounded to the left* if $\forall x \in W$, $\exists y \in W$ such that $y \precsim x$. These frame properties can be axiomatised by adding the following axiom(s):

$$\mathsf{G}\, p \supset \mathsf{F}\, p \qquad (\mathbf{D_r})$$

$$\mathsf{H}\, p \supset \mathsf{P}\, p \qquad (\mathbf{D}_\ell)$$

The logics we obtain when adding these axioms still have a finite model property and a small model property [Ono and Nakamura, 1980]. Moreover, a finite weak total order is unbounded to the right (resp. left) if and only if its rightmost (resp. leftmost) world is in a cluster. This leads to the following new rules for our calculus to handle unboundedness:

$$\frac{H \,;\, S \,;\, \{\vdash\}}{H \,;\, S}\ (\mathsf{D_r}) \qquad\qquad \frac{\{\vdash\} \,;\, S \,;\, H}{S \,;\, H}\ (\mathsf{D}_\ell)$$

Proposition 3.16. *Adding* ($\mathsf{D_r}$) *(resp.* (D_ℓ)*) yields a sound and complete proof system for* $\mathbf{K_t4.3} \cup (\mathbf{D_r})$ *(resp.* $\mathbf{K_t4.3} \cup (\mathbf{D}_\ell)$*), where proof search is in* coNP.

Proof. It is easy to check that rule ($\mathsf{D_r}$) is sound, as it reflects the shape of right-unbounded finite weak total orders. It is also invertible, since the underlying frame of the conclusion of the rule is always a subframe of its premises. Hence Lemma 3.7 and Lemma 3.5 still hold.

To obtain that proof search is in coNP, it suffices to check that Lemma 3.11 carries over to our extension. This is true because the rule ($\mathsf{D_r}$) can only be applied when the last cell of the hypersequent is not a cluster, and whenever the rule ($\mathsf{D_r}$) is applied, the last cell of its premises is always a cluster. Hence, the rule ($\mathsf{D_r}$) can only be applied once between applications of other rules creating new cells such as ($\vdash$G) and ($\vdash$H). Finally, since new cells can only be created polynomially many times by those other rules thanks to our initial strategy, the new rule ($\mathsf{D_r}$) can, in the end, only be applied polynomially many times along a branch. So the branches of our proof tree are still polynomial.

Finally, completeness is obtained as in Theorem 3.13. It only remains to show that the underlying frame of a failure hypersequent is unbounded to the right. Indeed, if its underlying frame was not unbounded to the right, we could apply the rule ($\mathsf{D_r}$) which would contradict the fact that no rules can be applied any more on this hypersequent. □

One can see that all rules can be taken together to form a sound and complete calculus for $\mathbf{K_tQ}$, with coNP proof search. Note that the rules proposed in this section differ from the ones proposed by Indrzejczak for density and unboundedness [Indrzejczak, 2016]. These rules would be sound but would break our polynomial bound on the length of proof branches.

3.4. First-Order Logic with Two Variables

We show here a coNEXP upper bound on the complexity of validity in the two-variable fragment of first-order logic over linear orders, re-proving and extending recent results by Manuel and Sreejith [2016].

Thanks to the translation result from Fact 2.7, we have therefore the following, where the NEXP upper bounds in items (i–iii) were already shown by Manuel and Sreejith [2016, Thm. 15] using automata-based techniques. Let us reiterate that the complexity bounds on the satisfiability problem for the modal logics in question were already known [Ono and Nakamura, 1980], so the interest here lies in the use of proof search in our hypersequent proof system rather than a brutal enumeration of all potential models up to some bound.

THEOREM 3.17. *The following problems are in* NEXP*: satisfiability of* $\mathrm{FO}^2(<)$ *over (i) arbitrary strict total orders, (ii) countable strict total orders, (iii) scattered strict total orders, and (iv) dense strict total orders.*

PROOF. Regarding (i), given an $\mathrm{FO}^2(<)$ formula ψ, we first turn it into the equisatisfiable formula $\exists y.\psi$ with one free variable x. Fact 2.7 then allows to construct a $\mathbf{K_t4.3}$ formula φ of exponential size, which is equisatisfiable over strict total orders. By Fact 2.6, it is also equisatisfiable over weak total orders, and Theorem 3.14 shows that satisfiability can be checked in non-deterministic polynomial time in $|\varphi|$, hence in NEXP overall.

Regarding (ii) and (iii), by Ono and Nakamura [1980, Thm. 3], the above-constructed φ is satisfiable over weak total orders if and only if it is satisfiable over finite weak total orders. The bulldozing construction used to prove Fact 2.6 (see Blackburn et al. [2001, Thm. 4.56]) consists essentially in turning each cluster into a direct product $\omega^* \cdot \omega$ (i.e., a copy of $\mathbb{Z}$), which shows that φ is satisfiable over finite weak total orders if and only if it is satisfiable over countable scattered strict total orders.

Finally, regarding (iv), by adapting Blackburn et al. [2001, theorems 4.41 and 4.56] to bulldoze clusters over $\mathbb{Q}$ rather than $\mathbb{Z}$, ψ is satisfiable over dense strict total orders if and only if the above-constructed φ is satisfiable over dense weak total orders as a $\mathbf{K_t4}.\cup$ (**Den**) formula. By Proposition 3.15, the latter can be checked in non-deterministic polynomial time in $|\varphi|$, hence in NEXP overall. □

3.5. Discussion

We have designed a sound and complete hypersequent calculus with clusters for the modal logic $\mathbf{K_t4.3}$ of linear temporal frames. The proof system takes advantage of the finite model property of our logic in the presence of clusters to bound the length of branches during a proof search, which yields a proof search with optimal coNP complexity for the validity problem. Moreover, the approach is modular, as these results remain true when extending the proof system to handle density and unboundedness, yielding a sound and complete system for $\mathbf{K_tQ}$ with the same complexity, and a sound and complete system for $\mathrm{FO}^2(<)$ with coNEXP upper bounds. This coNEXP upper bound itself is hardly surprising, but from a proof-theoretic perspective, the two-variable fragment of first-order logic is an unusual beast—eigenvariables must be avoided—, hence our solution through a proof system for a modal logic is arguably a natural one.

The system presented in this chapter is an improved version of the calculus from Baelde et al. [2018a] in which we introduced the notions of clusters and annotations. These were

inspired by the small model property of $\mathbf{K_t4.3}$ Ono and Nakamura [1980], and were more of an ad hoc way to implement model theoretic insights inside our proof system, whereas annotations are now seen as a new type of formulæ. This shift of perspective, together with the addition of rule ((G)), allows to get rid of the somewhat awkward use of different semantics for the soundness and completeness of the calculus from Baelde et al. [2018a]. It also frees the proof-theoretic development from the small model property; in fact, proof theory then allows to derive the small model property just as precisely.

In the next chapter, we investigate *well-founded* models, by adding the Gödel-Löb axiom to our logic. The models we will be working with are *ordinals*, and the newly obtained logic does not enjoy a finite model property. Nonetheless, we show how our hypersequent calculus can be extended to be sound and complete with respect to the tense logic over ordinals.

CHAPTER 4

Tense Logic over Ordinals

Contents

4.1. Introduction

We now focus on well-founded models. Linear temporal logic has become a staple specification language in verification since its introduction by Pnueli [1977]. In its most common form, the logic features an 'until' temporal modality and ranges over linear time flows of order type ω, i.e. over infinite words, where it enjoys a PSPACE-complete satisfiability problem [Sistla and Clarke, 1985]. A large number of variants with the same complexity has been motivated and introduced in the literature, notably temporal logics with past modalities [Laroussinie et al., 2002; Lichtenstein et al., 1985], ranging over arbitrary ordinals [Demri and Rabinovich, 2010; Rohde, 1997], or even—with the Stavi modalities added—over arbitrary linear time flows [Cristau, 2009; Rabinovich, 2012].

Linear temporal logic finds its roots in Prior's tense logic [Cocchiarella, 1965; Prior, 1957], which only featured the strict 'past' P and 'future' G modalities. This set of modalities is still interesting in its own right, as it is sufficient for many modelling tasks [Sistla and Zuck, 1993], and is known to lead to a slightly easier NP-complete satisfiability problem both over ω [Sistla and Clarke, 1985] and over arbitrary linear time flows [Ono and Nakamura, 1980]. While linear tense logic is less expressive than $\mathrm{FO}(<)$, the first-order logic over linear orders with unary predicates, it has nevertheless nice characterisations as it captures instead its two-variable fragment $\mathrm{FO}^2(<)$ [Etessami et al., 2002].

In this chapter, we investigate tense logic over well-founded linear time flows, i.e. over ordinals, which can be denoted as $\mathbf{K_tL}_\ell\mathbf{.3}$ in the taxonomy of modal logics from Blackburn et al. [2001]. We show in particular that

(1) the satisfiability problem for $\mathbf{K_tL}_\ell\mathbf{.3}$ over the class of ordinals is NP-complete, and that
(2) a formula φ of $\mathbf{K_tL}_\ell\mathbf{.3}$ has a well-founded linear model if and only if it has a model of order type α for some $\alpha < \omega \cdot (|\varphi| + 1)$; this should be contrasted with the corresponding $\omega^{|\varphi|+2}$ bound proven by Demri and Rabinovich [2010, Cor. 3.3] for linear temporal logic.

These two results are however just byproducts of our main contribution, which is a sound and complete proof system for $\mathbf{K_tL}_\ell\mathbf{.3}$ in which proof search runs in coNP. The hypersequent calculus presented in this chapter is obtained as a natural extension of our proof system for $\mathbf{K_t4.3}$ introduced in the previous chapter. The semantics of hypersequents has been adapted to now work with ordinals; and the rule ($\vdash$H) has been modified using additional insights from Avron's sequent calculus for **KL** [Avron, 1984]. This is satisfying since $\mathbf{K_tL}_\ell\mathbf{.3}$ is simply obtained from $\mathbf{K_t4.3}$—the tense logic of arbitrary linear time flows—by adding well-foundation to the left, i.e. towards the past (see Section 4.2), and completes the picture as $\mathbf{K_tQ}$ the tense logic of dense linear time flows was also handled in Chapter 3.

Furthermore, our proof system is easily shown in Section 4.5 to also address the more precise problems of validity over all the well-founded linear time flows

- of order type $\beta < \alpha$ for a given α, and
- of order type exactly $\alpha < \omega^2$.

Such a result seems out of reach of axiomatisations, and yields for instance a coNP decision procedure for validity over ω-words.

4.2. Tense Logic over Ordinals

4.2.1. Syntax. Our tense logic still has the same syntax, recalled below, the difference being the shape of the models we consider.

$$\varphi ::= \ \bot \mid p \mid \varphi \supset \varphi \mid \mathsf{G}\,\varphi \mid \mathsf{H}\,\varphi \qquad \text{(where } p \in \Phi\text{)}$$

Moreover, in order to guide the proof search, our calculus will still have to manipulate a different kind of future formulæ called *annotations*: these formulæ will be written $(\mathsf{G}\,\varphi)$, where $\mathsf{G}\,\varphi$ is a future modal formula, and will express that $\mathsf{G}\,\varphi$ holds starting from a specific later position. Remark that, as opposed to Chapter 3, we do not need past annotations any more, as the proof search will instead be guided by the well-foundedness of the models.

4.2.2. Ordinal Semantics. In the case of $\mathbf{K_tL}_\ell\mathbf{.3}$, our formulæ shall be evaluated on Kripke *structures* $\mathfrak{M} = (\alpha, V)$, where α is an ordinal and $V : \Phi \to \wp(\alpha)$ is a valuation of the propositional variables. Recall that an ordinal α is seen set-theoretically as $\{\beta \in \mathrm{Ord} \mid \beta < \alpha\}$. An ordinal is either 0 (the empty linear order), a *limit* ordinal λ (such that for all $\beta < \lambda$ there exists γ with $\beta < \gamma < \lambda$), or a *successor* ordinal $\alpha + 1$.

Given a structure $\mathfrak{M} = (\alpha, V)$, we define the *satisfaction* relation $\mathfrak{M}, \beta \models^{(\theta)} \varphi$, where $\beta < \alpha$, $\theta < \alpha$ and φ is a formula, by structural induction on φ. Notice that θ is only used for

the annotations.

$$\begin{aligned}
&\mathfrak{M}, \beta \not\models^{(\theta)} \bot \\
&\mathfrak{M}, \beta \models^{(\theta)} p && \text{iff } \beta \in V(p) \\
&\mathfrak{M}, \beta \models^{(\theta)} \varphi \supset \psi && \text{iff } \mathfrak{M}, \beta \models^{(\theta)} \varphi \text{ implies } \mathfrak{M}, \beta \models^{(\theta)} \psi \\
&\mathfrak{M}, \beta \models^{(\theta)} \mathsf{G}\,\varphi && \text{iff } \mathfrak{M}, \gamma \models^{(\theta)} \varphi \text{ for all } \gamma \text{ such that } \beta < \gamma \\
&\mathfrak{M}, \beta \models^{(\theta)} \mathsf{H}\,\varphi && \text{iff } \mathfrak{M}, \gamma \models^{(\theta)} \varphi \text{ for all } \gamma \text{ such that } \gamma < \beta \\
&\mathfrak{M}, \beta \models^{(\theta)} (\mathsf{G}\,\varphi) && \text{iff } \beta < \theta, \text{and } \mathfrak{M}, \gamma \models^{(\theta)} \varphi \text{ for all } \gamma \text{ such that } \theta \leq \gamma < \alpha
\end{aligned}$$

When $\mathfrak{M}, \beta \models^{(\theta)} \varphi$, we say that $(\mathfrak{M}, \beta)$ is a *model* of φ. Remark that, since annotations cannot appear as subformulæ, we have $\mathfrak{M}, \beta \models^{(\theta)} \varphi$ if and only if $\mathfrak{M}, \beta \models^{(\theta')} \varphi$ for any θ', when φ is not an annotation. In practice, when working with a formula $(\mathsf{G}\,\varphi)$, θ will correspond to a world to which the next cell is mapped, which will be either $\beta + \omega$ or $\beta + 1$, depending on whether $(\mathsf{G}\,\varphi)$ appears in a syntactic cluster or just a lone sequent.

Example 4.1. The satisfiable formulæ of $\mathbf{K_t L}_\ell\mathbf{.3}$ are strictly contained in the set of formulæ satisfiable in $\mathbf{K_t 4.3}$, i.e. over arbitrary linear orders. For instance, the formula $\varphi_0 = \mathsf{P}\,p \wedge \mathsf{H}\,(p \supset \mathsf{P}\,p)$ is satisfiable in $\mathbf{K_t 4.3}$ but not in $\mathbf{K_t L}_\ell\mathbf{.3}$, because all its models must contain an infinite decreasing sequence of worlds where p is true. Moreover, $\mathbf{K_t L}_\ell\mathbf{.3}$ can force models to be of order type greater than ω: for instance, the formula $\varphi_1 = \mathsf{G}\,(p \supset \mathsf{F}\,p) \wedge \mathsf{G}\,(\neg p \supset \mathsf{F}\,\neg p) \wedge \mathsf{F}\,\neg p \wedge \mathsf{F}\,(p \wedge \mathsf{G}\,p)$ forces to have a first infinite sequence of worlds not satisfying p, followed by a second infinite sequence of worlds satisfying p, and all its models (α, V) must have $\alpha \geq \omega \cdot 2$. Its smallest model is presented in Figure 4.1.

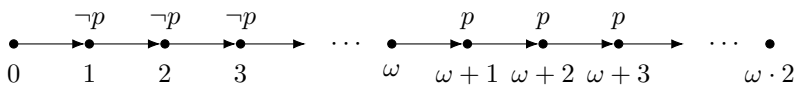

Figure 4.1. Minimal model of formula φ_1 from Example 4.1.

Following the spirit of φ_1, and using more propositional variables, it is actually possible to write a formula forcing its models to be of order type at least $\omega \cdot k + m$, for any k, m. However, our logic cannot express ordinals larger or equal to ω^2. This result comes from a small model property that can be deduced from the proof of completeness of our calculus (see Proposition 4.15).

4.3. Axiomatisation

For reference, the logic $\mathbf{K_t L}_\ell\mathbf{.3}$ can also be defined as the set of theorems generated by necessitation, modus ponens and substitution from classical tautologies and the axioms from $\mathbf{K_t 4.3}$ (cf. Section 2.4.3), along with the following axiom:

$$\mathsf{H}\,(\mathsf{H}\,\phi \supset \phi) \supset \mathsf{H}\,\phi \tag{$\mathbf{L}_\ell$}$$

This last axiom ($\mathbf{L}_\ell$), dubbed axiom of Gödel-Löb, ensures that the models are transitive and well-founded to the left.

4.4. Hypersequents with Clusters

We still use hypersequents with clusters presented in Section 3.2, but we adapt their semantics to work with ordinals.

4.4.1. Semantics. A *sequent* (denoted S) is still a pair of two finite sets of formulæ, written $\Gamma \vdash \Delta$. It is satisfied by worlds β and θ of a structure $\mathfrak{M}$ if the conjunction of the formulæ of Γ implies the disjunction of the formulæ of Δ. In that case, we write $\mathfrak{M}, \beta \models^{(\theta)} \Gamma \vdash \Delta$.

The semantics of an ordered hypersequent with clusters relies on a new notion of embedding which we define next, building on a view of hypersequents as partially ordered structures.

While a hypersequent is syntactically a finite partial order, its semantics will refer to a linear well-founded order, obtained by 'bulldozing' its clusters into copies of ω. The resulting order type is the object of the next definition.

Definition 4.2 (order type). Let H be a hypersequent. We define its *order type* $o(H)$ by induction on its structure: for cells, $o(\bullet) = 0$, $o(S) = 1$, and $o(\{Cl\}) = \omega$, and for hypersequents, $o(H_1 \,;\, H_2) = o(H_1) + o(H_2)$. Thus, $o(H) = \omega \cdot k + m$ where k is the number of clusters in H and m the number of non-empty cells to the right of the rightmost cluster.

Definition 4.3 (embedding). Let H be an annotated hypersequent and α an ordinal. We say that $\mu : \mathrm{dom}(H) \to \alpha + 1 \setminus \{0\}$ is an *embedding* of H into α, written $H \hookrightarrow_\mu \alpha$, if:

- for all $i, j \in \mathrm{dom}(H)$, $i \prec j$ implies $\mu(i) < \mu(j)$ and $i \sim j$ implies $\mu(i) = \mu(j)$; and
- for all $i \in \mathrm{dom}(H)$, i is in a cluster if and only if $\mu(i)$ is a limit ordinal.

Observe that, if $H \hookrightarrow_\mu \alpha$, then $o(H) < \alpha + 1$.

Definition 4.4 (semantics). A structure $\mathfrak{M}$ is a *model* of a hypersequent H if there exists an embedding $\mathfrak{M} \hookrightarrow_\mu H$, a position i of H, and an ordinal $\beta < \mu(i)$ such that, for all γ such that $\beta \leq \gamma < \mu(i)$, we have $\mathfrak{M}, \gamma \models^{(\mu(i))} H(i)$. In that case, we write $\mathfrak{M}, \mu \models H$.

Following this definition, we say that a hypersequent is *valid* if for any $\mathfrak{M} = (\alpha, V)$ and any embedding $H \hookrightarrow_\mu \mathfrak{M}$, $\mathfrak{M}, \mu \models H$. A formula φ is valid in the usual sense (i.e., satisfied in every world of every ordinal structure) if and only if the hypersequent $\vdash \varphi$ is valid in our sense.

If a hypersequent H is not valid, then it has a *counter-model*, that is a structure $\mathfrak{M} = (\alpha, V)$ and an embedding $H \hookrightarrow_\mu \mathfrak{M}$ such that for every $i \in \mathrm{dom}(H)$, for every $\beta < \mu(i)$, there exists γ with $\beta \leq \gamma < \mu(i)$ such that $\mathfrak{M}, \gamma \not\models^{(\mu(i))} H(i)$. For the positions $i \in \mathrm{dom}(H)$ that are not in clusters, $\mu(i)$ is a successor ordinal $\gamma + 1$ and this amounts to asking that $\mathfrak{M}, \gamma \not\models^{(\gamma+1)} H(i)$. When i is in a cluster, the condition implies the existence of an infinite increasing sequence $(\gamma_j)_j$ of ordinals with limit $\mu(i) = \sup_j \gamma_j$, such that $\mathfrak{M}, \gamma_j \not\models^{(\mu(i))} H(i)$ for all j.

4.4.2. Proof System. We now present our proof system for $\mathbf{K_t L}_\ell\mathbf{.3}$, called $\mathbf{HK_t L_\ell .3}$. The rules of this system consist of the rules of Chapter 3, presented in Figures 3.3 to 3.5, where the rule (⊢H) is replaced by the one presented in Figure 4.2. Since this new rule does not introduce past annotations any more, we also get rid of the rule ((H)).

The new rule (⊢H) works in the same spirit as the one from Figure 3.4, but does not rely on the syntactic clusters any more, as our models are now well-founded: these clusters now represent a substructure of order type ω and are only useful to deal with future modalities. Moreover, since the new rule (⊢H) never unboxes a formula inside a cluster, it does not need

$$\frac{\begin{array}{ll} H_2 \,;\, C' \,;\, \mathsf{H}\,\varphi \vdash \varphi \,;\, C\,[\Gamma \vdash \Delta, \mathsf{H}\,\varphi] \,;\, H_1 & \\ H_2 \,;\, C' \Subset (\vdash \mathsf{H}\,\varphi) \,;\, C\,[\Gamma \vdash \Delta, \mathsf{H}\,\varphi] \,;\, H_1 & \text{if } C' \neq \bullet \\ H_2 \,;\, C' \Subset (\mathsf{H}\,\varphi \vdash \varphi) \,;\, C\,[\Gamma \vdash \Delta, \mathsf{H}\,\varphi] \,;\, H_1 & \text{if } C' \neq \bullet \text{ and } C' \neq \{Cl\} \end{array}}{H_2 \,;\, C' \,;\, C\,[\Gamma \vdash \Delta, \mathsf{H}\,\varphi] \,;\, H_1}\ (\vdash\mathsf{H})$$

FIGURE 4.2. New rule ($\vdash$H) of $\mathbf{HK_tL_\ell.3}$. we allow $C' = \bullet$ only when $H_2 = \bullet$.

the use of past annotations any more, since $(\mathsf{H}\,\varphi)$ and $\mathsf{H}\,\varphi$ both have the same semantics when appearing in lone sequents.

EXAMPLE 4.5. The way annotations are handled differs from Baelde et al. [2018b]. As a result, the annotation rules from Baelde et al. [2018b] are subsumed by the version of the rule ((G)) presented in Figure 3.5. For instance, if H has a position i that is not in a cluster such that $H(i)$ contains $(\mathsf{G}\,\varphi) \vdash \mathsf{G}\,\varphi$, the branch can be immediately closed by some rule from Baelde et al. [2018b]. Let us show that such an H is provable by $\mathbf{HK_tL_\ell.3}$. First of all, since $(\mathsf{G}\,\varphi) \in H(i)$, then either $H(i)$ contains $(\mathsf{G}\,\varphi) \vdash \varphi$, or there exist $j \prec i$ such that $H(j)$ contains it. Then:

- If ($\vdash$G) cannot be applied on $\mathsf{G}\,\varphi$, it is either because $H(i+1)$ contains $(\mathsf{G}\,\varphi) \vdash \varphi$, and then H is immediately provable, or because $\mathsf{G}\,\varphi$ also appears on the right-hand side of $H(i+1)$, and the same formula can also be sent on its left-hand side (if not already present) by applying ((G)) on the annotation $(\mathsf{G}\,\varphi)$, and then (ax) can be used.
- Else, we apply ($\vdash$G) on $\mathsf{G}\,\varphi$. All premises are immediately provable, except for the premise sending $\mathsf{G}\,\varphi$ on the right-hand side of $H(i+1)$ which can be proved as in the previous case.

The rules of $\mathbf{HK_tL_\ell.3}$ are still designed to be *invertible*: by keeping in premises all the formulæ from the conclusion, we ensure that validity is never lost by applying a rule; this will be shown formally in Proposition 4.9. In practice, keeping all formulæ can be unnecessarily heavy. Fortunately, the rules (weak $\vdash$) and ($\vdash$ weak) from Section 3.2 are still admissible.

EXAMPLE 4.6. The formula $\varphi_0 = \mathsf{P}\,p \wedge \mathsf{H}\,(p \supset \mathsf{P}\,p)$ from Example 4.1 is not satisfiable in $\mathbf{K_tL_\ell.3}$, so the dual sequent $S_0 = \mathsf{H}\,(p \supset (\mathsf{H}\,(p \supset \bot) \supset \bot)) \vdash \mathsf{H}\,(p \supset \bot)$ is valid. Here is indeed a proof tree, with implicit uses of propositional and weakening rules, and principal formulæ shown in orange.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{H}\,(p \supset \bot), p \vdash p \,;\, S_0}\,(\text{ax}) \quad \dfrac{}{\mathsf{H}\,(p \supset \bot), p \vdash \mathsf{H}\,(p \supset \bot) \,;\, S_0}\,(\text{ax})}{p \supset (\mathsf{H}\,(p \supset \bot) \supset \bot), \mathsf{H}\,(p \supset \bot), p \vdash \,;\, S_0}\,(\supset \vdash)}{\mathsf{H}\,(p \supset \bot), p \vdash \,;\, \mathsf{H}\,(p \supset (\mathsf{H}\,(p \supset \bot) \supset \bot)) \vdash \mathsf{H}\,(p \supset \bot)}\,(\mathsf{H}\vdash)}{\mathsf{H}\,(p \supset (\mathsf{H}\,(p \supset \bot) \supset \bot)) \vdash \mathsf{H}\,(p \supset \bot)}\,(\vdash\mathsf{H})$$

EXAMPLE 4.7. Since $\varphi_1 = \mathsf{G}\,(p \supset \mathsf{F}\,p) \wedge \mathsf{G}\,(\neg p \supset \mathsf{F}\,\neg p) \wedge \mathsf{F}\,\neg p \wedge \mathsf{F}\,(p \wedge \mathsf{G}\,p)$ from Example 4.1 is satisfiable, its dual sequent $S_1 = \mathsf{G}\,(\mathsf{G}\,p \supset p), \mathsf{G}\,(p \supset (\mathsf{G}\,(p \supset \bot) \supset \bot)) \vdash \mathsf{G}\,p, \mathsf{G}\,\varphi$ where $\varphi = p \supset \bot \vee (\mathsf{G}\,p \supset \bot)$ is invalid, although with no counter-models below $\omega \cdot 2$.

In our calculus, proof search for S_1 will succeed on branches not considering at least two clusters; we show below in Figure 4.3 one such branch (with implicit uses of propositional and

weakening rules, and principal formulæ shown in orange). Its top-right sequent can be proved as explained in Example 4.5.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{S_1\,;\,(\mathsf{G}\,p),p\vdash p\,;\,\{(\mathsf{G}\,\varphi),\mathsf{G}\,p,p\vdash\}}\,(\mathsf{ax})\quad\dfrac{}{S_1\,;\,(\mathsf{G}\,p)\vdash p,\mathsf{G}\,p\,;\,\{(\mathsf{G}\,\varphi),\mathsf{G}\,p,p\vdash\}}}{S_1\,;\,(\mathsf{G}\,p),\mathsf{G}\,p\supset p\vdash p\,;\,\{(\mathsf{G}\,\varphi),\mathsf{G}\,p,p\vdash\}}\,(\supset\vdash)}{\mathsf{G}\,(\mathsf{G}\,p\supset p),\mathsf{G}\,(p\supset(\mathsf{G}\,(p\supset\bot)\supset\bot))\vdash\mathsf{G}\,p,\mathsf{G}\,\varphi\,;\,(\mathsf{G}\,p)\vdash p\,;\,\{(\mathsf{G}\,\varphi),\mathsf{G}\,p,p\vdash\}}\,(\mathsf{G}\vdash)\quad\ldots}\,(\vdash\mathsf{G})}{\mathsf{G}\,(\mathsf{G}\,p\supset p),\mathsf{G}\,(p\supset(\mathsf{G}\,(p\supset\bot)\supset\bot))\vdash\mathsf{G}\,p,\mathsf{G}\,(p\supset\bot\vee(\mathsf{G}\,p\supset\bot))}\,(\vdash\mathsf{G})$$

FIGURE 4.3. In a proof of S_1 (Example 4.7) branches with a non-cluster cell for $(\mathsf{G}\,p)$ are provable. On this branch, the top-right sequent can be proved as explained in Example 4.5.

However, proof search will fail on the branch shown in Figure 4.4, which corresponds to the counter-model described in Section 4.2.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{S_1\,;\,\{(\mathsf{G}\,p)\vdash\mathsf{G}\,p,p\}\,;\,\{(\mathsf{G}\,\varphi),\mathsf{G}\,p,p\vdash\|\,(\mathsf{G}\,(p\supset\bot)),p\vdash\mathsf{G}\,(p\supset\bot)\}}{S_1\,;\,\{(\mathsf{G}\,p)\vdash\mathsf{G}\,p,p\}\,;\,\{(\mathsf{G}\,\varphi),\mathsf{G}\,p,p\vdash\|\,(\mathsf{G}\,(p\supset\bot)),p,p\supset(\mathsf{G}\,(p\supset\bot)\supset\bot)\vdash\}}\,(\supset\vdash)}{S_1\,;\,\{(\mathsf{G}\,p)\vdash\mathsf{G}\,p,p\}\,;\,\{(\mathsf{G}\,\varphi),\mathsf{G}\,p,p\vdash\|\,(\mathsf{G}\,(p\supset\bot)),p\vdash\}}\,(\mathsf{G}\vdash)\quad\ldots}{S_1\,;\,\{(\mathsf{G}\,p)\vdash\mathsf{G}\,p,p\}\,;\,\{(\mathsf{G}\,\varphi),\mathsf{G}\,p,p\vdash\mathsf{G}\,(p\supset\bot)\}}\,(\vdash\mathsf{G})}{S_1\,;\,\{\mathsf{G}\,p\supset p,(\mathsf{G}\,p)\vdash p\}\,;\,\{p\supset(\mathsf{G}\,(p\supset\bot)\supset\bot),(\mathsf{G}\,\varphi),\mathsf{G}\,p,p\vdash\}}\,(\supset\vdash)\times 2}{\mathsf{G}\,(\mathsf{G}\,p\supset p),\mathsf{G}\,(p\supset(\mathsf{G}\,(p\supset\bot)\supset\bot))\vdash\mathsf{G}\,p,\mathsf{G}\,\varphi\,;\,\{(\mathsf{G}\,p)\vdash p\}\,;\,\{(\mathsf{G}\,\varphi),\mathsf{G}\,p,p\vdash\}}\,(\mathsf{G}\vdash)\times 2\quad\ldots}{\mathsf{G}\,(\mathsf{G}\,p\supset p),\mathsf{G}\,(p\supset(\mathsf{G}\,(p\supset\bot)\supset\bot))\vdash\mathsf{G}\,p,\mathsf{G}\,\varphi\,;\,\{(\mathsf{G}\,\varphi),\mathsf{G}\,p,p\vdash\}}\,(\vdash\mathsf{G})\quad\ldots}\,(\vdash\mathsf{G})$$

FIGURE 4.4. A failed branch in the proof of S_1 (Example 4.7).

4.4.3. Soundness.

PROPOSITION 4.8. *The rules of* $\mathbf{HK_tL_\ell.3}$ *are sound: if the premises of a rule instance are valid, then so is its conclusion.*

Even though most of the rules are the same, the semantics of hypersequents have evolved since last chapter, and we provide a new proof for the modal rules.

PROOF. We show the contrapositive: considering an application of a rule with a conclusion hypersequent H and a counter-model $(\mathfrak{M},\mu)$ of H with $\mathfrak{M}=(\alpha,V)$ and $H\hookrightarrow_\mu\alpha$ an embedding, we provide a counter-model of one of the premises (or a contradiction when there is no premise).

Since we will often have to extend an embedding with a value for a new position, we define $\mu+(i\mapsto\alpha)$ as the mapping μ' such that $\mu'(i)=\alpha$, $\mu'(k)=\mu(k)$ for $k<i$ and $\mu'(k+1)=\mu(k)$ for $k\geq i$ in the domain of μ.

The case of propositional rules (Figure 3.3) is immediate: The usual reasoning applies to the principal sequent, and the same embedding is used to obtain a counter-model of one of the premises.

Next we turn to the modal rules of Figure 3.4:

- Consider the case of ($\mathsf{G}\vdash$), applied with $\mathsf{G}\,\varphi, \Gamma \vdash \Delta$ at position i and $\Pi \vdash \Sigma$ at position j such that $i \precsim j$. Remark that the rule ensures that $i \neq j$, but we do not need this assumption to justify it. We show that $(\mathfrak{M}, \mu)$ is a counter-model of the premise H', concentrating on the only difference with H, at position j. For clarity we distinguish two cases:
  - When $i \prec j$, we also have $\mu(i) < \mu(j)$. Since $(\mathfrak{M}, \mu)$ is a counter-model of H, by taking an arbitrary $\beta_i < \mu(i)$ we obtain γ_i such that $\beta_i \leq \gamma_i < \mu(i)$ such that $\mathfrak{M}, \gamma_i \not\models^{(\mu(i))} H(i)$. In particular, $\mathfrak{M}, \gamma_i \models^{(\mu(i))} \mathsf{G}\,\varphi$. Now, considering an arbitrary $\beta < \mu(j)$ we need to exhibit γ such that $\beta \leq \gamma < \mu(j)$ and $\mathfrak{M}, \gamma \not\models^{(\mu(j))} H'(j)$. By taking $\beta_j = \max(\beta, \mu(i)) < \mu(j)$ we obtain γ_j such that $\beta_j \leq \gamma_j < \mu(j)$ and $\mathfrak{M}, \gamma_j \not\models^{(\mu(j))} H(j)$. Furthermore, since $\gamma_i < \mu(i) \leq \beta_j \leq \gamma_j$ and $\mathfrak{M}, \gamma_i \models^{(\mu(i))} \mathsf{G}\,\varphi$, we also have $\mathfrak{M}, \gamma_j \models^{(\mu(j))} \varphi$ and $\mathfrak{M}, \gamma_j \models^{(\mu(j))} \mathsf{G}\,\varphi$, hence $\mathfrak{M}, \gamma_j \not\models^{(\mu(j))} H'(j)$.
  - When $i \sim j$ we have that $\mu(i) = \mu(j)$ and it is a limit ordinal because we are considering positions in a cluster. Consider an arbitrary $\beta < \mu(i)$. There exists γ_i such that $\beta \leq \gamma_i < \mu(i)$ and $\mathfrak{M}, \gamma_i \not\models^{(\mu(i))} H(i)$. Because $\mu(i)$ is a limit ordinal, $\gamma_i + 1 < \mu(i) = \mu(j)$. Again, there exists γ_j such that $\gamma_i + 1 \leq \gamma_j < \mu(j)$ and $\mathfrak{M}, \gamma_j \not\models^{(\mu(j))} H(j)$. But, since $\gamma_i < \gamma_j$ we also have that γ_j satisfies φ and $\mathsf{G}\,\varphi$, hence $\mathfrak{M}, \gamma_j \not\models^{(\mu(j))} H'(j)$.
- The case of rule ($\{\mathsf{G}\vdash\}$) is covered by the second part of the previous argument, by taking $i = j$. Indeed, we have $i \sim i$ when ($\{\mathsf{G}\vdash\}$) applies at position i.
- Consider now an application of rule ($\mathsf{H}\vdash$) with $\Pi \vdash \Sigma$ at position i and $\mathsf{H}\,\varphi, \Gamma \vdash \Delta$ at j. We have $i \precsim j$, hence $\mu(i) \leq \mu(j)$. Consider an arbitrary $\beta < \mu(i)$. There exists γ_i such that $\beta \leq \gamma_i < \mu(i)$ and $\mathfrak{M}, \gamma_i \not\models^{(\mu(i))} H(i)$. We claim, as before, that there exists γ_j such that $\gamma_i < \gamma_j < \mu(j)$ and $\mathfrak{M}, \gamma_j \not\models^{(\mu(j))} H(j)$. Indeed, if $\mu(i) < \mu(j)$ then there exists γ_j with $\mu(i) \leq \gamma_j < \mu(j)$ that falsifies $H(j)$. Otherwise $\mu(i) = \mu(j)$ but then this must be a limit ordinal and, by considering $\gamma_i + 1 < \mu(i) = \mu(j)$ we obtain $\gamma_i < \gamma_j < \mu(j)$ that invalidates $H(j)$. Having $\mathfrak{M}, \gamma_j \not\models^{(\mu(j))} H(j)$, we also have $\mathfrak{M}, \gamma_j \models^{(\mu(j))} \mathsf{H}\,\varphi$. Thus γ_i satisfies φ and $\mathsf{H}\,\varphi$, and $\mathfrak{M}, \gamma_i \not\models^{(\mu(i))} H'(i)$ as needed.
- The case of ($\{\mathsf{H}\vdash\}$) is covered by the previous argument.
- Consider an application of ($\vdash\mathsf{G}$) with $\Gamma \vdash \Delta, \mathsf{G}\,\varphi$ at position i. For any $\beta_i < \mu(i)$ there exists γ_i with $\beta_i \leq \gamma_i < \mu(i)$ such that $\mathfrak{M}, \gamma_i \not\models^{(\mu(i))} H(i)$, and thus $\mathfrak{M}, \gamma_i \not\models^{(\mu(i))} \mathsf{G}\,\varphi$. Hence there also exists γ_i' with $\gamma_i < \gamma_i' < \alpha$ such that $\mathfrak{M}, \gamma_i' \not\models^{(\theta)} \varphi$ for any θ. Let γ be the least ordinal that is strictly larger than all such γ_i'. We have that $\mu(i) \leq \gamma$.

  We now distinguish several cases regarding γ. When $C'; H_2$ is not empty let j be the first position of the conclusion hypersequent that is in C'.
  - If $\mu(i) = \gamma$, then $\mu(i)$ must be a limit ordinal (for every $\beta_i < \mu(i)$, we can find $\beta_i < \gamma_i' < \gamma = \mu(i)$). Hence $C \neq \star$ and the third premise H_3' is available. We construct a counter-model $(\mathfrak{M}, \mu')$ for it by taking $\mu' = \mu + (k \mapsto \gamma)$, where $k = i + 1$ is the new position in H_3'. Indeed, we have that for any $\beta' < \mu'(k)$ there exists γ' with $\beta' \leq \gamma' < \mu'(k)$ and $\mathfrak{M}, \gamma' \not\models^{(\mu'(k))} \varphi$ (the inequality can even be made strict). Moreover, $\mathfrak{M}, \gamma' \models^{(\mu'(k))} (\mathsf{G}\,\varphi)$ by definition of $\gamma = \mu'(k)$: there cannot be any $\lambda \geq \gamma$ such that $\mathfrak{M}, \lambda \not\models^{(\gamma)} \varphi$.

- If C' ; H_2 is empty, or $\gamma < \mu(j)$, we conclude by observing that $(\mathfrak{M}, \mu')$ is a counter-model of one of the first two premises with $\mu' = \mu + (k \mapsto \gamma)$ where k is the position of the new cell in these premises. We check that μ' is monotone, because $\mu(i) < \gamma$, and $\gamma < \mu(j)$ when it is defined. If γ is a successor ordinal, $(\mathfrak{M}, \mu')$ is a counter-model of the first premise simply because the predecessor of γ invalidates φ and satisfies $(\mathsf{G}\,\varphi)$; both hold by construction. If γ is a limit ordinal we have a counter-model $(\mathfrak{M}, \mu')$ of the second premise: we do have that for any $\beta' < \mu'(k)$ there exists γ' with $\beta' \leq \gamma' < \mu'(k)$ that invalidates φ, and $(\mathsf{G}\,\varphi)$ is satisfied by construction.
- Otherwise $\mu(j) \leq \gamma$.
  - If $\mu(j) < \gamma$, we obtain a counter-model $(\mathfrak{M}, \mu)$ of the fourth premise H_4'. We check it for the only position whose sequent has changed between H and H_4', that is position j. Take any $\beta_j < \mu(j)$. We know that there exists γ_j with $\beta_j \leq \gamma_j < \mu(j)$ such that $\mathfrak{M}, \gamma_j \not\models^{(\mu(j))} H(j)$. But, since $\gamma_j < \mu(j) < \gamma$, there exists γ' such that $\gamma_j < \gamma' < \gamma$ and $\mathfrak{M}, \gamma' \not\models^{(\mu(j))} \varphi$. Thus $\mathfrak{M}, \gamma_j \not\models^{(\mu(j))} \mathsf{G}\,\varphi$, and $\mathfrak{M}, \gamma_j \not\models^{(\mu(j))} H_4'(j)$.
  - If $\mu(j) = \gamma$ and is a limit ordinal, we also obtain a counter-model $(\mathfrak{M}, \mu)$ of the fourth premise. This time, for any $\beta_j < \mu(j)$, we know that there exists γ_j with $\beta_j \leq \gamma_j < \mu(j)$ such that $\mathfrak{M}, \gamma_j \not\models^{(\mu(j))} H(j)$. But, since $\gamma_j < \gamma$ and γ is a limit ordinal, there still exists γ' such that $\gamma_j < \gamma' < \gamma$ and $\mathfrak{M}, \gamma' \not\models^{(\mu(j))} \varphi$. Thus $\mathfrak{M}, \gamma_j \not\models^{(\mu(j))} \mathsf{G}\,\varphi$, and $\mathfrak{M}, \gamma_j \not\models^{(\mu(j))} H_4'(j)$.
  - Finally, if $\mu(j) = \gamma$ and is not a limit ordinal, then the position j is not in a cluster, so the fifth premise is available. We claim that it admits $(\mathfrak{M}, \mu)$ as a counter-model. Let θ be the predecessor of $\gamma = \theta + 1$, which satisfies $\mathfrak{M}, \theta \not\models^{(\mu(j))} \varphi$ by definition of γ. Since $(\mathfrak{M}, \mu)$ is a counter-model of H we also have $\mathfrak{M}, \theta \not\models^{(\mu(j))} H(j)$. This allows us to conclude, together with the fact that, as before, the new annotation $(\mathsf{G}\,\varphi)$ is satisfied by definition of γ (there cannot be any $\lambda \geq \gamma$ such that $\mathfrak{M}, \lambda \not\models^{(\gamma)} \varphi$).

- We now consider an application of rule $(\vdash \mathsf{H})$ with $\Gamma \vdash \Delta, \mathsf{H}\,\varphi$ at position i. Let j be the first position of C', if it exists. For any $\beta_i < \mu(i)$ there exists γ_i with $\beta_i \leq \gamma_i < \mu(i)$ that invalidates $H(i)$, thus there exists $\gamma_i' < \gamma_i < \mu(i)$ such that $\mathfrak{M}, \gamma_i' \not\models^{(\mu(i))} \varphi$. Let γ be the successor of the least ordinal among all such γ_i'. We have $\gamma < \mu(i)$.
  - If $H_2; C'$ is empty, or $\mu(j) < \gamma$, then $(\mathfrak{M}, \mu')$ is a counter-model of the first premise with $\mu' = \mu + (k \mapsto \gamma)$ where k is the new position in that premise. We do have that the predecessor of γ satisfies $\mathsf{H}\,\varphi$ (by minimality) but not φ (by definition).
  - If $\mu(j) = \gamma$ then C' cannot be a cluster, because γ is a successor. In that case $(\mathfrak{M}, \mu)$ directly yields a counter-model of the third premise.
  - Otherwise $\gamma < \mu(j)$ and $(\mathfrak{M}, \mu)$ is a counter-model of the second premise.

We now consider the case of the future annotation rule from Figure 3.5. Consider an application of ((G)) with $(\mathsf{G}\,\varphi), \Gamma \vdash \Delta$ at position i and $\Pi \vdash \Sigma$ at position j, with $i \prec j$. Because of the annotation $(\mathsf{G}\,\varphi)$ we have that, for all $\lambda \geq \mu(i)$, $\mathfrak{M}, \lambda \models^{(\mu(i))} \varphi$. Hence $(\mathfrak{M}, \mu)$ is a counter-model of the premise. □

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{H}\,a, a \vdash a \,;\, \mathsf{H}\,b \vdash b \,;\, \mathsf{H}\,a \vdash a \,;\, \vdash \mathsf{H}\,a, \mathsf{H}\,b}\,(\mathsf{ax})}{\mathsf{H}\,a \vdash a \,;\, \mathsf{H}\,b \vdash b \,;\, \mathsf{H}\,a \vdash a \,;\, \vdash \mathsf{H}\,a, \mathsf{H}\,b}\,(\mathsf{H}\vdash) \quad \dfrac{b, \mathsf{H}\,a \vdash a \,;\, \mathsf{H}\,b \vdash b, \mathsf{H}\,a \,;\, \vdash \mathsf{H}\,a, \mathsf{H}\,b}{\mathsf{H}\,a \vdash a \,;\, \mathsf{H}\,b \vdash b, \mathsf{H}\,a \,;\, \vdash \mathsf{H}\,a, \mathsf{H}\,b}\,(\mathsf{H}\vdash) \quad \cdots}{\mathsf{H}\,a \vdash a \,;\, \mathsf{H}\,b \vdash b \,;\, \vdash \mathsf{H}\,a, \mathsf{H}\,b}\,(\vdash\mathsf{H}) \quad \cdots}{\mathsf{H}\,a \vdash a \,;\, \vdash \mathsf{H}\,a, \mathsf{H}\,b}\,(\vdash\mathsf{H})}{\vdash \mathsf{H}\,a, \mathsf{H}\,b}\,(\vdash\mathsf{H})$$

FIGURE 4.5. Proof search with a failure hypersequent and an immediately provable hypersequent.

4.4.4. Completeness and Complexity. As in Chapter 3, completeness is a by-product of the very simple proof-search behaviour of our calculus. As we shall see, all the rules are invertible and proof search branches are polynomially bounded, as long as obvious pitfalls are avoided in the search strategy. Thus it is useless to backtrack during proof-search. Moreover, proof attempts result in finite (polynomial depth) partial proofs, whose unjustified leaves yield counter-models that amount (by invertibility) to counter-models of the conclusion. Hence the completeness of our calculus. We detail this argument below, and its corollary: proof-search yields an optimal coNP procedure for validity.

PROPOSITION 4.9 (invertibility). *In any rule instance, if a premise has a counter-model, then so does its conclusion.*

PROOF. Considering a rule instance with a counter-model $(\mathfrak{M}, \mu)$ of a premise H, we build a counter-model $(\mathfrak{M}, \mu')$ of the conclusion H'. Just as in the proof of Lemma 3.5, we show that for any position i of H', for any β, $\mathfrak{M}, \beta \not\models^{(\mu(j))} H(j)$ implies $\mathfrak{M}, \beta \not\models^{(\mu'(i))} H'(i)$, where j is the corresponding position in H. □

As in Chapter 3, we use a notion of immediately provable hypersequents. This definition is updated to take into account the different behaviour of past navigation.

LEMMA 4.10. *If a hypersequent H satisfies one of these conditions, then H is provable.*

(a) There exists a formula φ, and two positions $i \prec j$ of H such that $H(i)$ and $H(j)$ both contain the sequent $(\mathsf{G}\,\varphi) \vdash \varphi$.

(b) There exists a formula φ, and two positions $i \prec j$ of H such that $H(i)$ and $H(j)$ both contain the sequent $\mathsf{H}\,\varphi \vdash \varphi$.

In such a case, we say that H is immediately provable.

PROOF. For every case, we show how to prove H.

(a) As in Lemma 3.8, such a hypersequent can still be proved as follows:

$$\dfrac{\dfrac{}{H_1\,[\Gamma, (\mathsf{G}\,\varphi) \vdash \varphi, \Delta] \,;\, H_2\,[\Gamma', (\mathsf{G}\,\varphi), \mathsf{G}\,\varphi, \varphi \vdash \varphi, \Delta']}\,(\mathsf{ax})}{H_1\,[\Gamma, (\mathsf{G}\,\varphi) \vdash \varphi, \Delta] \,;\, H_2\,[\Gamma', (\mathsf{G}\,\varphi) \vdash \varphi, \Delta']}\,((\mathsf{G}))$$

(b) This case is similar to Lemma 3.8, but uses $(\vdash\mathsf{H})$ instead of $((\mathsf{H}))$ as no past annotations are involved any more:

$$\dfrac{\dfrac{}{H_1\,[\Gamma, \mathsf{H}\,\varphi, \varphi \vdash \varphi, \Delta] \,;\, H_2\,[\Gamma', \mathsf{H}\,\varphi \vdash \varphi, \Delta']}\,(\mathsf{ax})}{H_1\,[\Gamma, \mathsf{H}\,\varphi \vdash \varphi, \Delta] \,;\, H_2\,[\Gamma', \mathsf{H}\,\varphi \vdash \varphi, \Delta']}\,(\vdash\mathsf{H})$$

□

We characterise next the proof attempts that we consider for proof search, and show how to extract counter-models when such attempts fail. We still use the notions of *partial proof* and *failure hypersequent* from Definition 3.9, but condition (iii) can now be ignored since no past annotations are involved any more. We show next that failure hypersequents are still invalid.

PROPOSITION 4.11. *Any failure hypersequent H has a counter-model.*

PROOF. Let $\alpha = o(H)$. We define $\mu : \mathrm{dom}(H) \to \alpha + 1 \setminus \{0\}$ as follows:

$$\begin{array}{ll} \mu(i) = m & \text{if } i \text{ is the } m\text{-th cell of } H \\ & \text{and appears before its first cluster;} \\ \mu(i) = \omega \cdot k & \text{if } i \text{ belongs to the } k\text{-th cluster of } H; \\ \mu(i) = \omega \cdot k + m & \text{if } i \text{ is the } m\text{-th cell appearing between} \\ & \text{the } k\text{-th and the next cluster (if any).} \end{array}$$

Now let $\mathrm{pos} : \alpha \to \mathrm{dom}(H)$ be a function such that:

(a) $\forall \beta < \beta' < \alpha,\ \mathrm{pos}(\beta) \precsim \mathrm{pos}(\beta')$
(b) $\forall \beta < \alpha,\ \forall i \in \mathrm{dom}(H),\ \beta < \mu(i) \Leftrightarrow (\mathrm{pos}(\beta) \precsim i \text{ or } \mathrm{pos}(\beta) = i)$
(c) $\forall \beta < \alpha,\ \forall i \in \mathrm{dom}(H),\ \mathrm{pos}(\beta) \precsim i \Rightarrow \exists \beta < \gamma < \mu(i), i = \mathrm{pos}(\gamma)$

There always exists one such function. Its choice is quite constrained due to the definitions of α and μ. Positions i that are not in a cluster will be such that $i = \mathrm{pos}(\beta)$ for a single β, typically the predecessor of $\mu(i)$. A position i appearing in a cluster must correspond to an infinite sequence of ordinals of limit $\mu(i)$, so that for all $i \sim j$ and β, if $\mathrm{pos}(\beta) = i$ then there exists γ with $\beta < \gamma < \mu(i) = \mu(j)$ such that $\mathrm{pos}(\gamma) = j$; informally, this ensures that positions i and j inside a cluster are 'infinitely interleaved' within $\mu(i) = \mu(j)$.

We finally define a valuation $V : \Phi \to \wp(\alpha)$ by

$$V(p) = \{\beta < \alpha \mid \exists \Gamma, \Delta\ .\ H(\mathrm{pos}(\beta)) = (p, \Gamma \vdash \Delta)\}$$

and let $\mathfrak{M} = (\alpha, V)$. We now claim that $\mathfrak{M}, \gamma \not\models^{(\mu(\mathrm{pos}(\gamma)))} H(\mathrm{pos}(\gamma))$ for all $\gamma < \alpha$: we prove by induction on ψ that, if ψ appears in the left-hand (resp. right-hand) side of the turnstile in $H(\mathrm{pos}(\gamma))$, then $\mathfrak{M}, \gamma \models^{(\mu(\mathrm{pos}(\gamma)))} \psi$ (resp. $\mathfrak{M}, \gamma \not\models^{(\mu(\mathrm{pos}(\gamma)))} \psi$).

- If ψ is an atom $p \in V$ the results follow by definition of V, and because (ax) does not apply to H. The propositional cases are obtained by induction hypothesis, because the corresponding rules of Figure 3.3 have already been applied.
- Assume that $\psi = (\mathsf{G}\,\varphi)$ appears on the left-hand side of the turnstile in $H(\mathrm{pos}(\gamma))$ (an annotation cannot appear on the right-hand side). Then, because ((G)) does not apply, φ appears on the left-hand side of $H(i)$ for any i such that $\mathrm{pos}(\gamma) \prec i$, so $\mathfrak{M}, \gamma' \models^{(\theta)} \varphi$ for every $\gamma' \geq \mu(\mathrm{pos}(\gamma))$ and for any θ, hence $\mathfrak{M}, \gamma \models^{(\mu(\mathrm{pos}(\gamma)))} (\mathsf{G}\,\varphi)$.
- The cases of modal formulæ on the left-hand side are similar, we only detail that of H. If $\psi = \mathsf{H}\,\varphi$ occurs on the left-hand side of $H(\mathrm{pos}(\gamma))$ then by ($\mathsf{H}\vdash$) and ($\{\mathsf{H}\vdash\}$), the formula φ must occur on the left-hand side of any $H(i)$ with $i \precsim \mathrm{pos}(\gamma)$. Moreover, for all $\gamma' < \gamma$, we have $\mathrm{pos}(\gamma') \precsim \mathrm{pos}(\gamma)$ by **(a)**, so $\mathfrak{M}, \gamma' \models^{(\mu(\mathrm{pos}(\gamma')))} \varphi$, and thus $\mathfrak{M}, \gamma \models^{(\mu(\mathrm{pos}(\gamma)))} \psi$.
- Assume that $\psi = \mathsf{H}\,\varphi$ occurs on the right of $H(\mathrm{pos}(\gamma))$. We prove by a sub-induction on $\mathrm{pos}(\gamma)$ that $\mathfrak{M}, \gamma \not\models^{(\mu(\mathrm{pos}(\gamma)))} \mathsf{H}\,\varphi$. Since ($\vdash\mathsf{H}$) does not apply, and since the first premise necessarily differs from the conclusion, it must be that there is a cell C'

preceding the cell that contains $\mathrm{pos}(\gamma)$, and that the last two premises (if available) would coincide with H. Let i be the first position in C'. Take an arbitrary $\lambda < \mu(i)$ such that $\mathrm{pos}(\lambda) = i$ (such a λ always exists, thanks to **(b)** and **(c)** instantiated with $\beta = 0$). Since $i \prec \mathrm{pos}(\gamma)$ it must be that $\lambda < \gamma$. As noted above, we have either that $\mathsf{H}\,\varphi$ belongs to the right-hand side of $H(i)$, or that φ belongs to its left-hand side. In the first case, we obtain $\mathfrak{M}, \lambda \not\models^{(\mu(\mathrm{pos}(\lambda)))} \mathsf{H}\,\varphi$ by induction hypothesis on $i < \mathrm{pos}(\gamma)$. In the second case we directly have $\mathfrak{M}, \lambda \not\models^{(\mu(\mathrm{pos}(\lambda)))} \varphi$. We conclude either way that $\mathfrak{M}, \gamma \not\models^{(\mu(\mathrm{pos}(\gamma)))} \mathsf{H}\,\varphi$.

- Assume finally that $\psi = \mathsf{G}\,\varphi$ occurs on the right-hand side of $H(\mathrm{pos}(\gamma))$. Let us first assume that there does not exist any $i \succ \mathrm{pos}(\gamma)$ such that $\mathsf{G}\,\varphi$ appears on the right-hand side of $H(i)$.
  - If ($\vdash$G) does not apply on ψ because of case (i) of condition (a), it cannot be because of the fourth premise by the previous assumption, so an annotation $(\mathsf{G}\,\varphi)$ must appear at some position i in H on the left-hand side of the turnstile, along with φ on its right-hand side. By rule ((G)) we must have $\mathrm{pos}(\gamma) \precsim i$ (or else (ax) could by applied). By **(c)**, there exists $\gamma' > \gamma$ such that $i = \mathrm{pos}(\gamma')$. We then have $\mathfrak{M}, \gamma' \not\models^{(\mu(\mathrm{pos}(\gamma')))} \varphi$, thus $\mathfrak{M}, \gamma \not\models^{(\mu(\mathrm{pos}(\gamma)))} \mathsf{G}\,\varphi$.
  - If ($\vdash$G) does not apply on ψ because of case (ii) of condition (a), there exists $i \sim \mathrm{pos}(\gamma)$ such that $H(i)$ contains $(\mathsf{G}\,\varphi) \vdash \varphi$, and we can conclude the same way we did above for case (i).

  Now, if there exists a position $i \succ \mathrm{pos}(\gamma)$ such that $\mathsf{G}\,\varphi$ appears on the right-hand side of $H(i)$, since we just proved that $\mathfrak{M}, \gamma' \not\models^{(\mu(i))} \mathsf{G}\,\varphi$ for any γ' with $\mathrm{pos}(\gamma') = i$ (in particular, $\gamma' > \gamma$), then we indeed have $\mathfrak{M}, \gamma \not\models^{(\mu(\mathrm{pos}(\gamma)))} \mathsf{G}\,\varphi$.

We can check that $H \hookrightarrow_\mu \alpha$: the conditions of Definition 4.3 hold by construction.

Finally, $(\mathfrak{M}, \mu)$ is a counter-model of H. Indeed, for all $i \in \mathrm{dom}(H)$ and $\beta < \mu(i)$ there exists γ with $\beta \leq \gamma < \mu(i)$ such that $\mathrm{pos}(\gamma) = i$, and hence $\mathfrak{M}, \gamma \not\models^{(\mu(i))} H(i)$: if $\mathrm{pos}(\beta) = i$, we can take $\gamma = \beta$, else **(b)** enforces $\mathrm{pos}(\beta) \precsim i$, and **(c)** provides one such γ. □

We now turn to establishing that proof search terminates, and always produces branches of polynomial length. The precise bound is the same as obtained in Lemma 3.11, and the proof works exactly the same.

Lemma 4.12 (small branch property). *For any partial proof of a hypersequent H, any branch of the proof is of length at most $2(|H| + \mathsf{len}(H) + 1) \cdot |H|$.*

We conclude that $\mathbf{HK_tL_\ell.3}$ is complete, and also enjoys optimal complexity proof search.

Theorem 4.13 (completeness). *Every valid hypersequent H has a proof in $\mathbf{HK_tL_\ell.3}$.*

Proof. Assume that H is not provable. Consider a partial proof $\mathcal{P}$ of H that cannot be expanded any more: its unjustified leaves are failure hypersequents. Such a partial proof exists by Lemma 4.12. Any unjustified leaf of that partial proof has a counter-model by Proposition 4.11, and by invertibility shown in Proposition 4.9 it is also a counter-model of H. □

Proposition 4.14. *Proof search in $\mathbf{HK_tL_\ell.3}$ is in* coNP.

Proof. Proof search can be implemented in an alternating Turing machine maintaining the current hypersequent on its tape, where existential states choose which rule to apply (and how) and universal states choose a premise of the rule. By Lemma 4.12, the computation

branches are of length bounded by a polynomial. By Proposition 4.9, the non-deterministic choices in existential states can be replaced by arbitrary deterministic choices, thus the resulting Turing machine has only universal states, hence is in coNP. □

4.5. Logic on Given Ordinals

We have designed a proof system that is sound and complete for $\mathbf{K_t L}_\ell\mathbf{.3}$, and enjoys optimal complexity proof search. We now show that this system can easily be enriched to obtain decision procedures not only for tense logic over arbitrary ordinals, but also for tense logic over specific ordinals. We first observe that the logic can only distinguish ordinals up to ω^2, which should be contrasted with Demri and Rabinovich [2010]. Then we show how to capture validity over ordinals below some $\omega \cdot k + m$, and finally how to reason over a specific ordinal of this form.

Proposition 4.15 (small model property). *If a hypersequent H has a counter-model, then it has a counter-model of order type $\alpha \leq \omega \cdot (|H| + \mathsf{len}(H))$.*

Proof. This is a corollary of Theorem 4.13. By the proof of Lemma 4.12, the hypersequents in a failure hypersequent—which are not immediately provable—have at most $|H| + \mathsf{len}(H)$ non-empty cells. The counter-model extracted in Proposition 4.11 from a failure hypersequent H' is over $o(H') \leq \omega \cdot (|H| + \mathsf{len}(H))$. A counter-model for H is then obtained by Proposition 4.9, with a different embedding but the same structure. □

In particular, for a formula φ, the hypersequent $H = \vdash \varphi$ has $|H| = |\varphi|$ and $\mathsf{len}(H) = 1$, hence the $\omega \cdot (|\varphi| + 1)$ bound announced in Section 4.1.

Next we observe that we can easily enrich our calculus to obtain a proof system for tense logic over ordinals below a certain type α.

Proposition 4.16. *Let α be an ordinal. The proof system $\mathbf{HK_t L}_\ell\mathbf{.3}$ enriched with the following axiom is sound and complete for tense logic over ordinals $\beta < \alpha$:*

$$\frac{}{H}\ (\mathsf{ord}_\alpha) \text{ if } o(H) \geq \alpha$$

Proof. The soundness argument for the rules of $\mathbf{HK_t L}_\ell\mathbf{.3}$ (Proposition 4.8) carries over to the restricted semantics, since the underlying structure (and ordinal) is never modified in the argument. Conversely, the completeness argument of Theorem 4.13 can be strengthened because, thanks to the new rule, we can guarantee that any failure hypersequent H is such that $o(H) < \alpha$, hence the extracted counter-model is also below this bound. □

Example 4.17. When extending $\mathbf{HK_t L}_\ell\mathbf{.3}$ to check for validity below ω, the failing branch of Figure 4.4 can be completed, as well as the other failing branches since they all involve hypersequents of order type $\omega \cdot 2$, and S_1 becomes provable.

We finally show how to capture validity at a fixed ordinal $\alpha < \omega^2$. The basic idea is to start with a hypersequent H such that $o(H) = \alpha = \omega \cdot k + m$ for some finite k and m, and take rule (ord_α) to forbid larger ordinals. The only catch is that we should check that the formula of interest is valid in all possible positions. Let us write $\{\vdash\}^k$ for $\{\vdash\}; \cdots ; \{\vdash\}$ with k clusters containing the empty sequent, and $(\vdash)^m$ for $\vdash; \cdots ; \vdash$ with m cells containing the empty sequent.

PROPOSITION 4.18. *The formula φ is valid in all structures of order type exactly $\alpha = \omega \cdot k + m$ if and only if* $\mathbf{HK_tL_\ell.3}$ *extended with* (ord_α) *proves all hypersequents of the form*

$$\{\vdash\}^{k_1} ; \vdash \varphi ; \{\vdash\}^{k_2} ; (\vdash)^m \quad \text{and} \quad \{\vdash\}^{k} ; (\vdash)^{m_1} ; \vdash \varphi ; (\vdash)^{m_2}$$

where $k_1 + k_2 = k$, $k_2 > 0$ and $m_1 + m_2 = m - 1$. In other words, one must consider all hypersequents H containing one sequent $\vdash \varphi$ and otherwise only empty sequents, and such that $o(H) = \omega \cdot k + m$.

For instance, when $k = m = 0$, φ vacuously holds in all worlds of $(0, V)$. When $k = 0$ and $m = 1$ we are checking $\vdash \varphi$ only, and (ord_α) closes any branch where a new cell is created, rendering modal formulæ trivially true. When $k = 1$ and $m = 0$ we are checking $\vdash \varphi ; \{\vdash\}$.

PROOF. If φ holds in all worlds of all structures of the form (α, V) for some V, the hypersequents are valid and thus provable in $\mathbf{HK_tL_\ell.3}$ with (ord_α). We prove the converse by contradiction. Assume that all the hypersequents hold and $\mathfrak{M}, \beta \not\Vdash \varphi$ for some $\mathfrak{M} = (\alpha, V)$ and $\beta < \alpha$. If $\omega \cdot k_1 \leq \beta < \omega \cdot (k_1 + 1)$ with $k_1 + 1 \leq k$ we can build an embedding to obtain a counter-model of the first kind of sequent. Otherwise, $\omega \cdot k \leq \beta < \omega \cdot k + m$ and we derive a counter-model of the second kind of sequent. □

EXAMPLE 4.19. Consider the formula $\mathrm{G}\varphi$ for $\varphi = \mathrm{G}\bot \supset \bot$. We cannot prove $\mathrm{G}\varphi$ in general, since this formula is not satisfied over finite ordinals, as witnessed by the following partial proof and its failure hypersequent (in the left branch) corresponding to a counter-model over the ordinal 2:

$$\dfrac{\dfrac{\vdash \mathrm{G}\varphi ; (\mathrm{G}\varphi), \mathrm{G}\bot \vdash \bot, \varphi}{\vdash \mathrm{G}\varphi ; (\mathrm{G}\varphi) \vdash \varphi}\,(\vdash \supset) \qquad \dfrac{\dfrac{\dfrac{}{\vdash \mathrm{G}\varphi ; \{(\mathrm{G}\varphi), \mathrm{G}\bot, \bot \vdash \bot, \varphi\}}\,(\bot)}{\vdash \mathrm{G}\varphi ; \{(\mathrm{G}\varphi), \mathrm{G}\bot \vdash \bot, \varphi\}}\,(\{\mathrm{G}\vdash\})}{\vdash \mathrm{G}\varphi ; \{(\mathrm{G}\varphi) \vdash \varphi\}}\,(\vdash \supset)}{\vdash \mathrm{G}\varphi}\,(\vdash \mathrm{G})$$

According to Proposition 4.18, over $\alpha = \omega$, i.e., $k = 1$ and $m = 0$, we need to prove $\vdash \mathrm{G}\varphi ; \{\vdash\}$ in $\mathbf{HK_tL_\ell.3}$ extended with (ord_ω), for which the presence of the cluster will be crucial. The extra rule (ord_ω) is actually not necessary in this case, but simplifies the proof. We start with an application of $(\vdash \mathrm{G})$, this time with three premises:

$$\dfrac{\vdash \mathrm{G}\varphi ; (\mathrm{G}\varphi) \vdash \varphi ; \{\vdash\} \quad \vdash \mathrm{G}\varphi ; \{(\mathrm{G}\varphi) \vdash \varphi\} ; \{\vdash\} \quad \vdash \mathrm{G}\varphi ; \{\vdash \mathrm{G}\varphi\}}{\vdash \mathrm{G}\varphi ; \{\vdash\}}\,(\vdash \mathrm{G})$$

The first premise is derived as follows:

$$\dfrac{\dfrac{\dfrac{}{\vdash \mathrm{G}\varphi ; (\mathrm{G}\varphi), \mathrm{G}\bot \vdash \bot, \varphi ; \{\bot \vdash\}}\,(\bot)}{\vdash \mathrm{G}\varphi ; (\mathrm{G}\varphi), \mathrm{G}\bot \vdash \bot, \varphi ; \{\vdash\}}\,(\mathrm{G}\vdash)}{\vdash \mathrm{G}\varphi ; (\mathrm{G}\varphi) \vdash \varphi ; \{\vdash\}}\,(\vdash \supset)$$

The middle premise can simply be discharged by (ord_ω). For the last premise, we use $(\vdash \mathrm{G})$ inside the cluster, which yields three premises: $\vdash \mathrm{G}\varphi ; \{\vdash \mathrm{G}\varphi\} ; (\mathrm{G}\varphi) \vdash \varphi$ and $\vdash \mathrm{G}\varphi ;$

$\{\vdash \mathsf{G}\,\varphi\}\,;\,\{(\mathsf{G}\,\varphi)\vdash\varphi\}$ are discharged by (ord_ω), while the last one is derived as follows:

$$\dfrac{\dfrac{\dfrac{}{\vdash \mathsf{G}\,\varphi\,;\,\{(\mathsf{G}\,\varphi),\bot\vdash \mathsf{G}\,\varphi \parallel \mathsf{G}\,\bot\vdash\bot,\varphi\}}\,(\bot)}{\vdash \mathsf{G}\,\varphi\,;\,\{(\mathsf{G}\,\varphi)\vdash \mathsf{G}\,\varphi \parallel \mathsf{G}\,\bot\vdash\bot,\varphi\}}\,(\mathsf{G}\vdash)}{\vdash \mathsf{G}\,\varphi\,;\,\{(\mathsf{G}\,\varphi)\vdash \mathsf{G}\,\varphi \parallel \vdash\varphi\}}\,(\vdash\supset)$$

4.6. Related Work and Conclusion

We have designed the first proof system for $\mathbf{K_tL}_\ell\mathbf{.3}$, i.e. tense logic over ordinals. Thanks to ordered hypersequents from Indrzejczak [2016], enriched with clusters and annotations as in Chapter 3, our system enjoys optimal complexity proof search, allows to derive small model properties, and can be extended into a proof system for variants of the logic over bounded or fixed ordinals.

Our $(\vdash\mathsf{H})$ rule is broadly related to the rule that Avron [1984] uses in his system for **KL**. Unlike Avron, we cannot work with standard sequents due to the presence of converse modalities. In turn, this allows us to consider a somewhat simpler right introduction rule for H, which does not have to take into account $\mathsf{H}\,\Gamma$ antecedents as they will remain available in the principal cell when a new one is created.

Finally, using the exponential translation of $\mathrm{FO}^2(<)$ into tense logic recalled in Section 2.4.4 [Etessami et al., 2002, Thm. 2], our results yield an optimal NEXP upper bound for satisfiability of the former over ordinals, which was already known from Otto [2001]. But more importantly they yield a proof system for $\mathrm{FO}^2(<)$ over ordinals, which would be challenging to construct directly, because eigenvariables cannot be handled in the usual fashion.

A natural next step would be to consider data words, infinite or not, which is the topic of the next chapter.

CHAPTER 5

Tense Logic over Data Ordinals

Contents

5.1. Introduction

Many applications can generate *data streams*, such as traces of a program's execution [Grigore et al., 2013], system logs [Bollig, 2011], or XML streams [Gauwin et al., 2011], which motivated the introduction of several *data logics* able to formally reason about such streams.

One of the first such logic is quite naturally the First Order Logic on data words, where a binary predicate allows to test whether two positions carry the same datum; however it is necessary to consider the two variable fragment [Bojańczyk et al., 2011] to get a decidable logic. Various other logics have been extended with a way to work with data. Pnueli's Linear Temporal Logic [Pnueli, 1977] is the base of the Logic of Repeated Values [Demri et al., 2012, 2016], and of Freeze LTL [Demri and Lazić, 2009; Figueira and Segoufin, 2009; Lazić, 2011] which uses the freeze quantifiers introduced by Alur and Henzinger [Alur and Henzinger, 1994], the modal μ-calculus is extended by the Fixpoint Logic on data words [Colcombet and Manuel, 2014], and PDL [Fischer and Ladner, 1979] is the backbone of XPath [Figueira, 2012a,b; Figueira and Segoufin, 2009, 2017], which allows to work either with data words or data trees.

However, as for the First Order Logic, such data logics are often undecidable, and with fragments of high complexity: the decidable fragments of freeze LTL proposed in [Demri and Lazić, 2009; Figueira and Segoufin, 2009] and various fragments of XPath [Figueira, 2012b; Figueira and Segoufin, 2009, 2017] are not primitive recursive, and the fragment from [Figueira, 2012a] is EXP-complete. Furthermore, not many of these works have considered the case of infinite words [Bojańczyk et al., 2011; Bollig, 2011; Colcombet and Manuel, 2014; Lazić, 2011],

even though it is the natural way to model system logs or XML streams. Moreover, working more generally with data ordinals instead would allow to study such problems more accurately.

In this chapter, we investigate the freeze tense logic over ordinals, which we call $\mathbf{K_t^{\downarrow}L}_\ell\mathbf{.3}$, and which combines freeze quantifiers [Alur and Henzinger, 1994; Demri and Lazić, 2009] with the logic $\mathbf{K_tL}_\ell\mathbf{.3}$ introduced in the previous chapter. This logic is known to be undecidable even with only one register, as the corresponding logic over finite words [Demri and Lazić, 2009], and even when considering only 'simple' formulæ [Figueira and Segoufin, 2009] where the use of that single register is restricted. In Section 5.4, we present a decidable fragment of $\mathbf{K_t^{\downarrow}L}_\ell\mathbf{.3}$, dubbed $\mathbf{K_t^dL}_\ell\mathbf{.3}$, in which the use of registers are even more restricted than in 'simple' formulæ from Figueira and Segoufin [2009]. This fragment is quite natural, as it is exactly as expressive as the two-variable fragment of first-order logic over data ordinals from Bojańczyk et al. [2011] (cf. Section 5.6). We show in particular that

(1) the satisfiability problem for $\mathbf{K_t^dL}_\ell\mathbf{.3}$ over the class of ordinals is NP-complete, and that
(2) a formula φ of $\mathbf{K_t^dL}_\ell\mathbf{.3}$ has a well-founded linear model if and only if it has a model of order type α for some $\alpha < \omega \cdot (4 \cdot |\varphi|^2 + |\varphi| + 2)$; this should be contrasted with the corresponding $\omega \cdot (|\varphi| + 1)$ bound proven in Chapter 4 for the underlying data-free logic $\mathbf{K_tL}_\ell\mathbf{.3}$.

These two results are however just by-products of our main contribution, which is a sound and complete proof system for $\mathbf{K_t^dL}_\ell\mathbf{.3}$ in which proof search runs in coNP.

Our proof system for $\mathbf{K_t^dL}_\ell\mathbf{.3}$ is obtained as a natural extension of the previous chapter's proof system for $\mathbf{K_tL}_\ell\mathbf{.3}$, using additional rules to deal with registers, and strategy to make sure that proof search always produces proof attempts of polynomial depth. This is satisfying since $\mathbf{K_t^dL}_\ell\mathbf{.3}$ is a fragment of $\mathbf{K_t^{\downarrow}L}_\ell\mathbf{.3}$, which is simply obtained from $\mathbf{K_tL}_\ell\mathbf{.3}$—the tense logic over ordinals—by adding freeze quantifiers. Conceptually, re-using the framework required to slightly adapt it to work with data ordinals.

Furthermore, the results of Section 4.5 still apply to $\mathbf{HK_t^dL}_\ell\mathbf{.3}$: our proof system is easily shown in Section 5.5 to also address the more precise problems of validity over all the data ordinals

- of order type $\beta < \alpha$ for a given α, and
- of order type exactly $\alpha < \omega^2$.

Such a result seems out of reach for axiomatisations, and yields for instance a coNP decision procedure for validity over infinite data words.

5.2. Freeze Tense Logic over Ordinals

We present the freeze tense logic over ordinals, combining the Tense Logic over ordinals from Chapter 4 and freeze quantifiers.

5.2.1. Syntax. Our logic, called $\mathbf{K_t^{\downarrow}L}_\ell\mathbf{.3}$, features the same two unary temporal operators from tense logic, and countably many freeze operators, over a countable set Φ of propositional variables, with the following syntax:

$$\varphi ::= \ \bot \mid p \mid \varphi \supset \varphi \mid \mathsf{G}\,\varphi \mid \mathsf{H}\,\varphi \mid \downarrow_r \varphi \mid \uparrow_r \qquad \text{(where } p \in \Phi \text{ and } r \in \mathbb{N}\text{)}$$

Formulæ $\downarrow_r \varphi$ are called *freeze formulæ*, and atoms $\uparrow_r$ are called *thaw formulæ*. Intuitively, given a register r, $\downarrow_r \varphi$ stores the datum of the current world in the register r, and evaluates φ, and $\uparrow_r$ tests if the current world has the same datum as the one stored in the register r. Any occurrence of a thaw $\uparrow_r$ within the scope of a freeze quantifier $\downarrow_r$ is bound by it; otherwise, that thaw is *free*.

Furthermore, in order to guide the proof search, our calculus will still rely on future annotations of the form $(\mathsf{G}\,\varphi)$.

5.2.2. Data Ordinal Semantics. In the case of $\mathbf{K}_{\mathbf{t}}^{\downarrow}\mathbf{L}_{\ell}.\mathbf{3}$, our formulæ shall be evaluated on *data ordinals*, which are tuples (α, δ) with α an ordinal and δ a function mapping each element from α to a datum from an infinite[1] domain $\mathbb{D}$. Models of our logic are Kripke *structures* $\mathfrak{M} = (\mathfrak{F}, V, \nu)$, where the frame $\mathfrak{F} = (\alpha, \delta)$ is a data ordinal, $V : \Phi \to \wp(\alpha)$ is a valuation of the propositional variables, and ν is a finite partial map from $\mathbb{N}$ to $\mathbb{D}$ called a *register valuation*. The domain of such a ν must contain all the free registers that appear in the formulæ it evaluates.

Given a structure $\mathfrak{M} = ((\alpha, \delta), V)$ and a register valuation ν, we define the *satisfaction* relation $\mathfrak{M}, \beta \models_{\nu}^{(\theta)} \varphi$, where $\beta < \alpha$, $\theta < \alpha$ and φ is a formula, by structural induction on φ. Ordinals β and θ play the same roles as in the previous chapter, and ν is only involved in the semantics of freeze formulæ.

$$
\begin{array}{ll}
\mathfrak{M}, \beta \not\models_{\nu}^{(\theta)} \bot & \\
\mathfrak{M}, \beta \models_{\nu}^{(\theta)} p & \text{iff } \beta \in V(p) \\
\mathfrak{M}, \beta \models_{\nu}^{(\theta)} \varphi \supset \psi & \text{iff } \mathfrak{M}, \beta \models_{\nu}^{(\theta)} \varphi \text{ implies } \mathfrak{M}, \beta \models_{\nu}^{(\theta)} \psi \\
\mathfrak{M}, \beta \models_{\nu}^{(\theta)} \mathsf{G}\,\varphi & \text{iff } \mathfrak{M}, \gamma \models_{\nu}^{(\theta)} \varphi \text{ for all } \gamma \text{ such that } \beta < \gamma \\
\mathfrak{M}, \beta \models_{\nu}^{(\theta)} \mathsf{H}\,\varphi & \text{iff } \mathfrak{M}, \gamma \models_{\nu}^{(\theta)} \varphi \text{ for all } \gamma \text{ such that } \gamma < \beta \\
\mathfrak{M}, \beta \models_{\nu}^{(\theta)} \downarrow_r \varphi & \text{iff } \mathfrak{M}, \beta \models_{\nu[r \mapsto \delta(\beta)]}^{(\theta)} \varphi \\
\mathfrak{M}, \beta \models_{\nu}^{(\theta)} \uparrow_r & \text{iff } \delta(\beta) = \nu(r) \\
\mathfrak{M}, \beta \models_{\nu}^{(\theta)} (\mathsf{G}\,\varphi) & \text{iff } \beta < \theta, \text{and } \mathfrak{M}, \gamma \models_{\nu}^{(\theta)} \varphi \text{ for all } \gamma \text{ such that } \theta \leq \gamma < \alpha
\end{array}
$$

When $\mathfrak{M}, \beta \models_{\nu}^{(\theta)} \varphi$, we say that $(\mathfrak{M}, \nu, \beta, (\theta))$ is a *model* of φ. As in Chapter 4, since annotations cannot appear as subformulæ, we have $\mathfrak{M}, \beta \models_{\nu}^{(\theta)} \varphi$ if and only if $\mathfrak{M}, \beta \models_{\nu}^{(\theta')} \varphi$ for any θ', when φ is not an annotation.

Moreover, we note $[x/y](\varphi)$ for the formula φ where every free occurrence of the register y is replaced by the register x. More formally, we define it by structural induction as follows:

$$
\begin{array}{rlrl}
[x/y](\bot) &= \bot & [x/y](p) &= p \\
[x/y](\mathsf{G}\,\varphi) &= \mathsf{G}\,[x/y](\varphi) & [x/y](\mathsf{H}\,\varphi) &= \mathsf{H}\,[x/y](\varphi) \\
[x/y]((\mathsf{G}\,\varphi)) &= (\mathsf{G}\,[x/y](\varphi)) & [x/y](\varphi_1 \supset \varphi_2) &= [x/y](\varphi_1) \supset [x/y](\varphi_2) \\
[x/y](\uparrow_r) &= \uparrow_r \text{ if } r \neq y & [x/y](\uparrow_y) &= \uparrow_x
\end{array}
$$

[1] Since we will only be able to perform equality tests between data values, we can assume that $\mathbb{D}$ is countable.

$$[x/y](\downarrow_r \varphi) = \downarrow_r [x/y](\varphi) \text{ if } r \neq y \text{ and } r \neq x$$
$$[x/y](\downarrow_x \varphi) = \downarrow_r [x/y]([r/x](\varphi)) \text{ where } r \text{ is fresh}$$
$$[x/y](\downarrow_y \varphi) = \downarrow_y \varphi$$

Then, the following substitution lemma holds:

LEMMA 5.1. *For every formula φ, and every model $(\mathfrak{M}, \nu, \beta, (\theta))$ of φ:*

- *if $\nu(x) = \nu(y)$, then $\mathfrak{M}, \beta \models^{(\theta)}_{\nu} [x/y](\varphi)$.*
- *if no free occurrence of $\uparrow_x$ appears in φ, then $\mathfrak{M}, \beta \models^{(\theta)}_{\nu[x \mapsto \nu(y)]} [x/y](\varphi)$.*

Even though the underlying data-free logic $\mathbf{K_t L}_\ell\mathbf{.3}$ cannot express that a model is of order type at least ω^2 (cf. Proposition 4.15), this can be done with $\mathbf{K}^{\downarrow}_{\mathbf{t}}\mathbf{L}_\ell\mathbf{.3}$, even without using any propositional variable, as shown in the next example.

EXAMPLE 5.2. Let $\varphi_1 = \mathsf{G}(\downarrow_r \mathsf{F} \uparrow_r)$, $\varphi_2 = \mathsf{G}(\downarrow_r \mathsf{F} \neg \uparrow_r)$, and $\varphi_3 = \mathsf{G}(\downarrow_r \mathsf{G}(\mathsf{F} \uparrow_r \supset \uparrow_r))$. Then, $\varphi = \mathsf{F}\top \wedge \varphi_1 \wedge \varphi_2 \wedge \varphi_3$ is satisfiable, and any model of φ is of order type at least ω^2.

Because of $\mathsf{F}\top$, the other formulæ do not quantify over an empty set of future positions: there exists at least a future β_1. φ_1 forces that every datum appears infinitely many times, and φ_3 forces that every such infinite sequence of positions carrying the same datum is continuous (two such sequences for two different data cannot be interleaved). Hence, a smallest model of φ starts with ω positions carrying $d_1 = \delta(\beta_1)$. Finally, φ_2 forces the existence of β_2 carrying a datum d_2 such that $d_1 \neq d_2$. Because of φ_3, such a β_2 must be after that first ω carrying d_1; and because of φ_2 we must have a second ω carrying the datum d_2 at each position. Now, φ_2 forces the existence of β_3 carrying d_3 different from d_1 and d_2, we a third ω carrying d_3 must exist. In the end, by repeating this reasoning, a smallest model of φ must possess ω positions carrying the datum d, for infinitely many $d \in \mathbb{D}$, so is of size at least ω^2.

Moreover, φ is indeed satisfied by a model of size ω^2, where the ith ω carries d_i, for an enumeration $(d_i)_{i \in \mathbb{N}}$ of $\mathbb{D}$.

5.3. Hypersequents with Clusters

We still use the same hypersequents as before, but we now need to generalise their semantics to work with *data* ordinals.

In Section 5.3.2, we present new rules to add to $\mathbf{HK_t L_\ell.3}$ to handle freeze formulæ. The newly obtained calculus, dubbed $\mathbf{HK}^{\mathbf{d}}_{\mathbf{t}}\mathbf{L}_{\boldsymbol{\ell}}\mathbf{.3}$, is sound for for $\mathbf{K}^{\downarrow}_{\mathbf{t}}\mathbf{L}_\ell\mathbf{.3}$, as we will show in Section 5.3.3. However, since $\mathbf{K}^{\downarrow}_{\mathbf{t}}\mathbf{L}_\ell\mathbf{.3}$ is undecidable, our usual proof of completeness cannot work for the whole logic. In Section 5.4, we focus on a decidable fragment of $\mathbf{K}^{\downarrow}_{\mathbf{t}}\mathbf{L}_\ell\mathbf{.3}$, dubbed $\mathbf{K}^{\mathbf{d}}_{\mathbf{t}}\mathbf{L}_\ell\mathbf{.3}$, and prove that our calculus is complete for that fragment, and has a proof strategy of optimal complexity.

5.3.1. Semantics. As expected, a *sequent* of the form $\Gamma \vdash \Delta$ is satisfied by worlds β and θ of a structure $\mathfrak{M}$ if there exists a register valuation ν such that $\mathfrak{M}, \beta \models^{(\theta)}_{\nu} \bigwedge \Gamma \supset \bigvee \Delta$. We still use the same notions of embedding, partial order and order type of an hypersequent.

DEFINITION 5.3 (semantics). A structure $\mathfrak{M}$ is a *model* of a hypersequent H if there exists a register valuation ν, an embedding $\mathfrak{M} \hookrightarrow_\mu H$, and a position i of H such that for all $d \in \mathbb{D}$

$$\frac{\begin{array}{ll} H\,[\Gamma, \downarrow_r \varphi, \uparrow_x, [x/r](\varphi) \vdash \Delta] & \text{if } \forall y \in \mathbb{N}, \uparrow_y \notin \Gamma, \text{ with } x \text{ fresh} \\ H\,[\Gamma, \downarrow_r \varphi, [x/r](\varphi) \vdash \Delta] & \text{if } \uparrow_x \text{ is the only thaw atom in } \Gamma \end{array}}{H\,[\Gamma, \downarrow_r \varphi \vdash \Delta]}\ (\downarrow \vdash)$$

$$\frac{\begin{array}{ll} H\,[\Gamma, \uparrow_x \vdash [x/r](\varphi), \downarrow_r \varphi, \Delta] & \text{if } \forall y \in \mathbb{N}, \uparrow_y \notin \Gamma, \text{ with } x \text{ fresh} \\ H\,[\Gamma \vdash [x/r](\varphi), \downarrow_r \varphi, \Delta] & \text{if } \uparrow_x \text{ is the only thaw atom in } \Gamma \end{array}}{H\,[\Gamma \vdash \downarrow_r \varphi, \Delta]}\ (\vdash \downarrow)$$

$$\frac{[x/y](H\,[\uparrow_x, \Gamma \vdash \Delta])}{H\,[\uparrow_x, \uparrow_y, \Gamma \vdash \Delta]}\ (\uparrow \vdash)$$

FIGURE 5.1. Freeze, and thaw rules of $\mathbf{HK^d_t L_\ell.3}$. By fresh, we mean that x does not appear as a free register anywhere in the conclusion.

there exists an ordinal $\beta_d < \mu(i)$ such that for all γ such that $\beta_d \leq \gamma < \mu(i)$ and $\delta(\gamma) = d$, we have $\mathfrak{M}, \gamma \models_\nu^{(\mu(i))} H(i)$. In that case, we write $\mathfrak{M}, \nu, \mu \models H$.

Following this definition, we say that a hypersequent is *valid* if for any $\mathfrak{M} = ((\alpha, \delta), V)$, any embedding $H \hookrightarrow_\mu \mathfrak{M}$ and any register valuation ν, $\mathfrak{M}, \nu, \mu \models H$. A formula φ is valid in the usual sense (i.e., satisfied in every world of every ordinal structure) if and only if the hypersequent $\vdash \varphi$ is valid in our sense.

If a hypersequent H is not valid, then it has a *counter-model*, that is a structure $\mathfrak{M} = ((\alpha, \delta), V)$, an embedding $H \hookrightarrow_\mu \mathfrak{M}$ and a register valuation ν such that for every $i \in \mathrm{dom}(H)$ there exists $d_i \in \mathbb{D}$ such that for every $\beta < \mu(i)$, there exists γ with $\beta \leq \gamma < \mu(i)$ and $\delta(\gamma) = d_i$ such that $\mathfrak{M}, \gamma \not\models_\nu^{(\mu(i))} H(i)$. For the positions $i \in \mathrm{dom}(H)$ that are not in clusters, $\mu(i)$ is a successor ordinal $\gamma + 1$ and this amounts to asking that $\mathfrak{M}, \gamma \not\models_\nu^{(\gamma+1)} H(i)$. When i is in a cluster, the condition implies the existence of an infinite increasing sequence $(\gamma_j)_j$ of ordinals carrying the same datum, and with limit $\mu(i) = \sup_j \gamma_j$, such that $\mathfrak{M}, \gamma_j \not\models_\nu^{(\mu(i))} H(i)$ for all j.

5.3.2. Proof System. We now present our proof system for $\mathbf{K^\downarrow_t L}_\ell\mathbf{.3}$, called $\mathbf{HK^d_t L_\ell.3}$. It contains the rules from $\mathbf{HK_t L_\ell.3}$ (see Figures 3.3 to 3.5 and 4.2), to which we add the rules from Figure 5.1 to now handle freeze formulæ. In these rules, $[x/y](H)$ stands for the hypersequent H where the operator $[x/y]$ has been applied to every formula.

The rule $(\uparrow \vdash)$ unifies two registers when they must contain the same datum, and is helpful to bound the number of registers appearing in the proof search. The rules $(\downarrow \vdash)$ and $(\vdash \downarrow)$ both handle the freeze quantifier $\downarrow_r$ by adding a version of φ where r has been replaced by either an already used register matching the current datum if any, or a fresh one otherwise.

Our rules are still *invertible*: validity is never lost by applying a rule; this will be shown formally in Proposition 5.5. Moreover, the rules (weak $\vdash$) and ($\vdash$ weak) are still admissible. This may not seem obvious since the new rules $(\downarrow \vdash)$ and $(\vdash \downarrow)$ require some specific checks. To prove this claim, once the original proof system is proven complete, one could just prove that the weakening rules are sound. Nonetheless, we will sometimes omit formulæ when they

do not play any role to lighten some examples. Every time we do so, the exact same proof could be derived without omitting any formulæ.

5.3.3. Soundness.

PROPOSITION 5.4. *The rules of* $\mathbf{HK^d_t L_\ell.3}$ *are sound: if the premises of a rule instance are valid, then so is its conclusion.*

PROOF. We show the contrapositive: considering an application of a rule with a conclusion hypersequent H and a counter-model $(\mathfrak{M}, \nu, \mu)$ of H with $\mathfrak{M} = (\alpha, V)$ and $H \hookrightarrow_\mu \alpha$ an embedding, we provide a counter-model of one of the premises (or a contradiction when there is no premise).

The cases of the rules from Chapter 4 work exactly the same, so only the new rules from Figure 5.1 need to be considered:

- Consider an application of $(\downarrow \vdash)$ with $H(i) = \Gamma, \downarrow_r \varphi \vdash \Delta$. If there exists $\uparrow_x \in \Gamma$, then $(\mathfrak{M}, \nu, \mu)$ is also a counter-model of the premise. Else, since $(\mathfrak{M}, \nu, \mu)$ is a counter-model of H, there exists $d_i \in \mathbb{D}$ such that for all $\beta < \mu(i)$ there exists γ such that $\beta \leq \gamma < \mu(i)$, $\delta(\gamma) = d_i$, and $\mathfrak{M}, \gamma \not\models^{(\mu(i))}_\nu H(i)$. In particular, $\mathfrak{M}, \gamma \models^{(\mu(i))}_\nu \downarrow_r \varphi$, so $\mathfrak{M}, \gamma \models^{(\mu(i))}_{\nu[r \mapsto d_i]} \uparrow_r \wedge \varphi$. Let us take $\nu' = \nu[x \mapsto d_i]$. Since x is chosen fresh, it is also the case that $\mathfrak{M}, \gamma \not\models^{(\mu(i))}_{\nu'} H(i)$; and by the second case of Lemma 5.1, $\mathfrak{M}, \gamma \models^{(\mu(i))}_{\nu'} \uparrow_x \wedge [x/r](\varphi)$. Hence, $(\mathfrak{M}, \nu', \mu)$ is a counter-model of the premise.
- The case of $(\vdash \downarrow)$ is similar.
- Consider the application of $(\uparrow \vdash)$ with $H(i) = \uparrow_x, \uparrow_y, \Gamma \vdash \Delta$. Since there exists γ such that $\mathfrak{M}, \gamma \models^{(\mu(i))}_\nu \uparrow_x$ and $\mathfrak{M}, \gamma \models^{(\mu(i))}_\nu \uparrow_y$, then $\nu(x) = \nu(y) = \delta(\gamma)$. Hence, by the first case of Lemma 5.1, $(\mathfrak{M}, \nu, \mu)$ is a counter-model of the premise. □

PROPOSITION 5.5 (invertibility). *In any rule instance, if a premise has a counter-model, then so does its conclusion.*

PROOF. Considering a rule instance with a counter-model $(\mathfrak{M}, \nu, \mu)$ of a premise H, we build a counter-model $(\mathfrak{M}, \nu', \mu')$ of the conclusion H'. Just like for Lemma 3.5, we construct a proper embedding μ' of H' into $\mathfrak{M}$. Moreover, except for the rule $(\uparrow \vdash)$, all the free $\uparrow_r$ of H' are also free registers of H, so taking $\nu' = \nu$ suffices in these cases. For the case of $(\uparrow \vdash)$, H' must have two formulæ $\uparrow_x$ and $\uparrow_y$ on the left-hand side of some sequent such that $\uparrow_y$ is not a subformula of H, and we can take $\nu' = \nu[y \mapsto \nu(x)]$. It is then easy to see that $(\mathfrak{M}, \nu', \mu')$ is a counter-model of H', since any sequent $H'(i)$ is contained in the corresponding sequent $H(j)$ (up to some register renaming for the rule $(\uparrow \vdash)$): for any β, $\mathfrak{M}, \beta \not\models^{(\mu(j))}_\nu H(j)$ implies $\mathfrak{M}, \beta \not\models^{(\mu'(i))}_{\nu'} H'(i)$. □

As we have seen multiple times in this part, invertibility can be useful to prove completeness if we can prove that proof search always terminates. However, $\mathbf{K^\downarrow_t L_\ell.3}$ being undecidable, we now investigate a decidable fragment for which $\mathbf{HK^d_t L_\ell.3}$ is complete and has a proof strategy of optimal complexity.

5.4. Restricted Logic and Completeness

The previous logic is known to be undecidable, even with only one register [Demri and Lazić, 2009] and some restrictions regarding the use of that register [Figueira and Segoufin,

2009]. Here, we consider another restriction of the logic, and prove that our calculus is complete for that fragment, with proof search in coNP.

5.4.1. Restricted Syntax. We consider the following fragment of $\mathbf{K}_\mathbf{t}^{\downarrow}\mathbf{L}_\ell.\mathbf{3}$, that we call $\mathbf{K}_\mathbf{t}^{\mathbf{d}}\mathbf{L}_\ell.\mathbf{3}$:

$$\begin{aligned}\varphi ::= {} & \bot \mid p \mid \varphi \supset \varphi \mid \mathsf{G}\,\varphi \mid \mathsf{H}\,\varphi \\ & \mid \downarrow_r \mathsf{G}\,(\uparrow_r \supset \varphi) \mid \downarrow_r \mathsf{G}\,(\neg\uparrow_r \supset \varphi) \\ & \mid \downarrow_r \mathsf{H}\,(\uparrow_r \supset \varphi) \mid \downarrow_r \mathsf{H}\,(\neg\uparrow_r \supset \varphi) \qquad \text{(where } p \in \Phi \text{ and } r \in \mathbb{N}\text{)}\end{aligned}$$

Because the use of registers is restricted to such specific formulæ, we define the following syntactic sugar:

$$\mathsf{G}_{=r}\,\varphi = \mathsf{G}\,(\uparrow_r \supset \varphi) \qquad \mathsf{G}_{\neq r}\,\varphi = \mathsf{G}\,(\neg\uparrow_r \supset \varphi)$$
$$\mathsf{H}_{=r}\,\varphi = \mathsf{H}\,(\uparrow_r \supset \varphi) \qquad \mathsf{H}_{\neq r}\,\varphi = \mathsf{H}\,(\neg\uparrow_r \supset \varphi)$$

Intuitively, $\mathsf{G}_{=r}\,\varphi$ (resp. $\mathsf{G}_{\neq r}\,\varphi$) expresses the fact that φ holds in every future position with the same (resp. a different) datum as the one stored in the register r; and $\mathsf{H}_{=r}\,\varphi$, $\mathsf{H}_{\neq r}\,\varphi$ express the same for past positions. Moreover, since a negation before a freeze quantifier can be moved inside its scope, e.g. $\neg\downarrow_r \mathsf{G}_{\neq r}\,\neg\varphi \equiv \downarrow_r \neg \mathsf{G}_{\neq r}\,\neg\varphi$, we can also define their dual diamond modalities, e.g. $\mathsf{F}_{\neq r}\,\varphi = \neg\mathsf{G}_{\neq r}\,\neg\varphi$. Formulæ of the form $\downarrow_r \mathsf{G}_{=r}\,\varphi$ (resp. $\downarrow_r \mathsf{G}_{\neq r}\,\varphi$) corresponds to formulæ denoted $\Box_{=}\varphi$ (resp. $\Box_{\neq}\varphi$) from Baelde et al. [2016], which works over data trees.

Example 5.6. The formula from Example 5.2, forcing its models to have order type at least ω^2, does not belong to this fragment. Furthermore, there is no equivalent formula belonging to $\mathbf{K}_\mathbf{t}^{\mathbf{d}}\mathbf{L}_\ell.\mathbf{3}$, as we will show later that satisfiable formulæ from this fragment always have a model of order type strictly below ω^2.

From now on, we only consider formulæ from $\mathbf{K}_\mathbf{t}^{\mathbf{d}}\mathbf{L}_\ell.\mathbf{3}$.

5.4.2. Completeness and Complexity. As in the other chapters of this part, completeness is a by-product of a rather simple proof-search strategy. As already stated in Proposition 5.5, all the rules are invertible; and as we shall see, our strategy only produces proof trees with branches that are polynomially bounded for the restricted logic, as it will avoid any pitfall that could happen. Thus it is unnecessary to backtrack during proof-search. Moreover, proof attempts result in finite (polynomial depth) partial proofs, whose unjustified leaves yield counter-models that amount (by invertibility) to counter-models of the conclusion. Hence the completeness of our calculus. We detail this argument below, and its corollary: proof-search yields an optimal coNP procedure for validity.

We characterise next the proof attempts that we consider for proof search, and show how to extract counter-models when such attempts fail. Our strategy is unchanged for formulæ that belong to $\mathbf{K}_\mathbf{t}\mathbf{L}_\ell.\mathbf{3}$, but freeze formulæ must be handled more carefully.

Lemma 5.7. *If a hypersequent H satisfies one of these conditions, then H is provable.*

(a) There exists a formula φ, and two positions $i \prec j$ of H such that $H(i)$ and $H(j)$ both contain the sequent $(\mathsf{G}\,\varphi) \vdash \varphi$.

(b) There exists a formula φ, and two positions $i \prec j$ of H such that $H(i)$ and $H(j)$ both contain the sequent $\mathsf{H}\,\varphi \vdash \varphi$.

(c) *There exists a formula φ, three positions $i \prec j \prec k$ of H, and three registers $x, y, z \in \mathbb{N}$ such that:*
- *$H(i)$ contains $(\mathsf{G}_{\neq x}\,\varphi) \vdash \neg\uparrow_x \supset \varphi$.*
- *$H(j)$ contains $(\mathsf{G}_{\neq y}\,\varphi) \vdash \neg\uparrow_y \supset \varphi$.*
- *$H(k)$ contains $(\mathsf{G}_{\neq z}\,\varphi) \vdash \neg\uparrow_z \supset \varphi$.*

(d) *There exists a formula φ, three positions $i \prec j \prec k$ of H, and three registers $x, y, z \in \mathbb{N}$ such that:*
- *$H(i)$ contains $\mathsf{H}_{\neq x}\,\varphi \vdash \neg\uparrow_x \supset \varphi$.*
- *$H(j)$ contains $\mathsf{H}_{\neq y}\,\varphi \vdash \neg\uparrow_y \supset \varphi$.*
- *$H(k)$ contains $\mathsf{H}_{\neq z}\,\varphi \vdash \neg\uparrow_z \supset \varphi$.*

In such a case, we say that H is immediately provable.

Proof. Cases (a) and (b) work as in Lemma 4.10. We handle here the new cases.

(c) Let us first establish the following: if there is a formula φ and a register r such that a sequent of H contains $\neg\uparrow_r \supset \varphi \vdash \varphi$, then we can make $\uparrow_r$ appears on the left-hand side of this sequent:

$$\dfrac{\dfrac{}{H\,[\Gamma, \varphi \vdash \varphi, \Delta]}\,(\mathsf{ax}) \quad \dfrac{H\,[\Gamma, \neg\uparrow_r \supset \varphi, \uparrow_r \vdash \varphi, \Delta]}{H\,[\Gamma, \neg\uparrow_r \supset \varphi \vdash \neg\uparrow_r, \varphi, \Delta]}}{H\,[\Gamma, \neg\uparrow_r \supset \varphi \vdash \varphi, \Delta]}\,(\supset \vdash)$$

We now sketch in Figure 5.2 how to prove a hypersequent satisfying condition (c). In order to make the figure more readable, we omit other formulæ that could appear at positions i, j or k. The proof would work the same in presence of additional formulæ. The omitted steps correspond to the ones described above, for registers x and y. The last hypersequent satisfies condition (a), hence it is provable.

(d) This case is similar to (c), with the roles of positions i and k reverted (as well as roles of registers x and z), and using ($\vdash\mathsf{H}$) instead of ((G)); and reducing to an instance of (b). □

$$\dfrac{\dfrac{\dfrac{[x/y](H_1)\,[(\mathsf{G}_{\neq x}\,\varphi) \vdash \neg\uparrow_x \supset \varphi]\,;\,[x/y](H_2)\,[(\mathsf{G}_{\neq x}\,\varphi) \vdash \neg\uparrow_x \supset \varphi]\,;\,[x/y](H_3)\,[(\mathsf{G}_{\neq z}\,\varphi), \uparrow_x, \neg\uparrow_z \vdash \varphi]}{H_1\,[(\mathsf{G}_{\neq x}\,\varphi) \vdash \neg\uparrow_x \supset \varphi]\,;\,H_2\,[(\mathsf{G}_{\neq y}\,\varphi) \vdash \neg\uparrow_y \supset \varphi]\,;\,H_3\,[(\mathsf{G}_{\neq z}\,\varphi), \uparrow_x, \uparrow_y, \neg\uparrow_z \vdash \varphi]}\,(\uparrow \vdash)}{\vdots}}{H_1\,[(\mathsf{G}_{\neq x}\,\varphi) \vdash \neg\uparrow_x \supset \varphi]\,;\,H_2\,[(\mathsf{G}_{\neq y}\,\varphi) \vdash \neg\uparrow_y \supset \varphi]\,;\,H_3\,[(\mathsf{G}_{\neq x}\,\varphi), (\mathsf{G}_{\neq y}\,\varphi), (\mathsf{G}_{\neq z}\,\varphi), \neg\uparrow_x \supset \varphi, \neg\uparrow_y \supset \varphi, \neg\uparrow_z \vdash \varphi]}\,(\vdash \supset)}{H_1\,[(\mathsf{G}_{\neq x}\,\varphi) \vdash \neg\uparrow_x \supset \varphi]\,;\,H_2\,[(\mathsf{G}_{\neq y}\,\varphi) \vdash \neg\uparrow_y \supset \varphi]\,;\,H_3\,[(\mathsf{G}_{\neq x}\,\varphi), (\mathsf{G}_{\neq y}\,\varphi), (\mathsf{G}_{\neq z}\,\varphi), \neg\uparrow_x \supset \varphi, \neg\uparrow_y \supset \varphi \vdash \neg\uparrow_z \supset \varphi]}\,((\mathsf{G}))}{H_1\,[(\mathsf{G}_{\neq x}\,\varphi) \vdash \neg\uparrow_x \supset \varphi]\,;\,H_2\,[(\mathsf{G}_{\neq y}\,\varphi) \vdash \neg\uparrow_y \supset \varphi]\,;\,H_3\,[(\mathsf{G}_{\neq x}\,\varphi), (\mathsf{G}_{\neq z}\,\varphi), \neg\uparrow_x \supset \varphi \vdash \neg\uparrow_z \supset \varphi]}\,((\mathsf{G}))$$

Figure 5.2. Proof tree sketch for a hypersequent satisfying condition (c).

The intuition behind (d) is the following: if there exists γ where $\mathsf{H}_{\neq z}\,\varphi$ holds, and $\gamma' < \gamma$ where $\mathsf{H}_{\neq y}\,\varphi$ holds, and if y and z stores different data, then φ holds in every past position of γ' (at any position, either $\neg\uparrow_z$ or $\neg\uparrow_y$ holds) and thus any $\mathsf{H}_{\neq x}\,\varphi$ holds in the past. The intuition behind (c) is similar. This reasoning fails if y and z store the same datum, but we can always assume otherwise during proof search unless when $\uparrow_y$ and $\uparrow_z$ appear on the left-hand side of the same sequent, and in this case we should apply ($\uparrow \vdash$) in priority.

We characterise next the proof attempts that we consider for proof search, and show how to extract counter-models when such attempts fail.

Definition 5.8. We call *partial proof* a finite derivation tree whose internal nodes correspond to rule applications, but whose leaves may be unjustified hypersequents, and that satisfies three conditions:

(a) No rule application should be such that, if H is the conclusion hypersequent,
  (i) one of the premises is also H, or
  (ii) the rule being applied is ($\vdash$G) on a formula $\mathsf{G}\,\varphi$ at position i such that there exists $j \sim i$ such that $H(j)$ contains $(\mathsf{G}\,\varphi) \vdash \varphi$, or
  (iii) the rule being applied is ($\vdash$G) on a formula $\mathsf{G}\,(\neg\uparrow_x \supset \varphi)$ at position i such that there exists $j \sim i$ and $y \neq x$ such that $H(j)$ contains $\big(\mathsf{G}\,(\neg\uparrow_y \supset \varphi)\big) \vdash \neg\uparrow_y \supset \varphi$ and does not contain $\uparrow_x$ on its left-hand side.

(b) Immediately provable hypersequents must be proven immediately as described in the proof of Lemma 5.7.

(c) Else, if the rule ($\uparrow \vdash$) is applicable, or if the rule ($\vdash \supset$) is applicable on a formula of the form $\uparrow_x \supset \varphi$, then the other rules cannot be applied.

Finally, we call *failure hypersequent* a hypersequent on which any rule application would not respect condition (a).

The second part of condition (c) is there to optimise the use of its first part, which in turn is there to bound the number of registers our calculus manipulates during a proof search. As in the previous chapters, conditions (a) and (b) amount to a simple proof search strategy that avoids loops, and addresses especially loops arising from repeated applications of ($\vdash$H) or ($\vdash$G), in branches where several new cells are created for the same modal formula (up to maybe a different register): this results either in immediately provable hypersequents from Lemma 5.7, or failure hypersequents on which the proof strategy is stuck and for which we prove next that we can always construct a counter-model.

Proposition 5.9. *Any failure hypersequent H has a counter-model.*

Proof. The proof works as in Proposition 4.11, but we now need to also handle data. Let $\alpha = o(H)$. We still define $\mu : \mathrm{dom}(H) \to \alpha + 1 \setminus \{0\}$ as follows:

$$\begin{array}{ll} \mu(i) = m & \text{if } i \text{ is the } m\text{-th cell of } H \\ & \text{and appears before its first cluster;} \\ \mu(i) = \omega \cdot k & \text{if } i \text{ belongs to the } k\text{-th cluster of } H; \\ \mu(i) = \omega \cdot k + m & \text{if } i \text{ is the } m\text{-th cell appearing between} \\ & \text{the } k\text{-th and the next cluster (if any).} \end{array}$$

Moreover, we still manipulate a function $\mathrm{pos} : \alpha \to \mathrm{dom}(H)$ such that:

(a) $\forall \beta < \beta' < \alpha,\ \mathrm{pos}(\beta) \precsim \mathrm{pos}(\beta')$
(b) $\forall \beta < \alpha,\ \forall i \in \mathrm{dom}(H),\ \beta < \mu(i) \Leftrightarrow (\mathrm{pos}(\beta) \precsim i \text{ or } \mathrm{pos}(\beta) = i)$
(c) $\forall \beta < \alpha,\ \forall i \in \mathrm{dom}(H),\ \mathrm{pos}(\beta) \precsim i \Rightarrow \exists \beta < \gamma < \mu(i),\ i = \mathrm{pos}(\gamma)$

In addition, we now define the data assignment δ of α. Since the rule ($\uparrow \vdash$) cannot be applied on H, each position of H must have at most one atomic formula of the form $\uparrow_r$ on its left-hand side. For each position i of H, we choose a datum $d_i \in \mathbb{D}$ with the following constraints:

- If $H(i)$ and $H(j)$ have the same atomic $\uparrow_r$ on their left-hand side, then $d_i = d_j$;
- Else, $d_i \neq d_j$.

We can now define δ by $\delta(\beta) = d_{\mathrm{pos}(\beta)}$, for all $\beta < \alpha$. From this, we fix a fresh datum $d_\perp$ different from all the d_i, and we can define a register valuation ν defined for every free register that appears in H by:

- $\nu(r) = d_i$ if $\uparrow_r$ appears on the left-hand side of $H(i)$,
- $\nu(r) = d_\perp$ otherwise.

Finally, we still define the valuation $V : \Phi \to \wp(\alpha)$ by

$$V(p) = \{\beta < \alpha \mid \exists \Gamma, \Delta \,.\, H(\mathrm{pos}(\beta)) = (p, \Gamma \vdash \Delta)\}.$$

Let $\mathfrak{M} = ((\alpha, \delta), V)$. We now claim that $\mathfrak{M}, \gamma \not\models_\nu^{(\mu(\mathrm{pos}(\gamma)))} H(\mathrm{pos}(\gamma))$ for all $\gamma < \alpha$: we prove by induction on ψ that, if ψ appears in the left-hand (resp. right-hand) side of the turnstile in $H(\mathrm{pos}(\gamma))$, then $\mathfrak{M}, \gamma \models_\nu^{(\mu(\mathrm{pos}(\gamma)))} \psi$ (resp. $\mathfrak{M}, \gamma \not\models_\nu^{(\mu(\mathrm{pos}(\gamma)))} \psi$).

- If ψ is a thaw atom $\uparrow_r$, then:
  - If ψ appears on the left-hand side of the turnstile, the results follows by definition of ν.
  - If ψ appears on the right-hand side of the turnstile at position i, then either ψ also appears on the left-hand side of some position j, and $i \neq j$ because (ax) does not apply, so $\nu(r) = d_j \neq d_i$; or ψ never appears on the left-hand side of some sequent of H and $\nu(r) = d_\perp \neq d_i$. Either way, $\mathfrak{M}, \gamma \not\models_\nu^{(\mu(i))} \uparrow_r$ for γ such that $\mathrm{pos}(\gamma) = i$.
- Because $(\downarrow \vdash)$ and $(\vdash \downarrow)$ do not apply on H, there is no formula of the form $\downarrow_r \psi'$ anywhere in H for which $(\downarrow \vdash)$ or $(\vdash \downarrow)$ has not been applied (such rules could also be prevented by having two formulæ $\uparrow_x$ and $\uparrow_y$ on the left-hand side of a sequent, but this cannot happen here since $(\uparrow \vdash)$ does not apply). This means that every such $\downarrow_r \psi'$ appears along with $[x/r](\psi')$ on the same side of the turnstile, and $\uparrow_x$ on the left-hand side of the turnstile, for some x. Let us assume that $\downarrow_r \psi'$ and $[x/r](\psi')$ appear on the right-hand side of $H(\mathrm{pos}(\gamma))$ (the other case is similar). By induction hypothesis, $\mathfrak{M}, \gamma \models_\nu^{(\mu(\mathrm{pos}(\gamma)))} \uparrow_x$ and $\mathfrak{M}, \gamma \not\models_\nu^{(\mu(\mathrm{pos}(\gamma)))} [x/r](\psi')$, hence $\mathfrak{M}, \gamma \not\models_\nu^{(\mu(\mathrm{pos}(\gamma)))} \downarrow_r \psi'$.

All the other cases works exactly as in Proposition 4.11, since these cases do not manipulate data, and we can check that $H \hookrightarrow_\mu \alpha$: the conditions of Definition 4.3 hold by construction.

Finally, $(\mathfrak{M}, \nu, \mu)$ is a counter-model of H. Indeed, for all $i \in \mathrm{dom}(H)$ and $\beta < \mu(i)$ there exists γ with $\beta \leq \gamma < \mu(i)$ such that $\mathrm{pos}(\gamma) = i$, and hence $\mathfrak{M}, \gamma \not\models_\nu^{(\mu(i))} H(i)$: if $\mathrm{pos}(\beta) = i$, we can take $\gamma = \beta$, else **(b)** enforces $\mathrm{pos}(\beta) \precsim i$, and **(c)** provides one such γ. □

We now turn to establishing that proof search terminates, and always produces branches of polynomial length. For a hypersequent H, let $\mathsf{len}(H)$ be its number of sequents (i.e., the size of $\mathrm{dom}(H)$), and $|H|$ the number of distinct subformulæ occurring in H.

Lemma 5.10 (small branch property). *For any partial proof of a hypersequent H, any branch of the proof is of length at most* $2 \cdot |H| \cdot (4 \cdot |H| + \mathsf{len}(H)) \cdot ((4 \cdot |H| + \mathsf{len}(H)) \cdot |H| + \mathsf{len}(H) + 1)$.

Proof. Let H be a hypersequent, $\mathcal{P}$ a partial proof of it, and B a branch of $\mathcal{P}$. We note Φ_H the set of subformulæ of H. Remark that all the formulæ that appear in B belongs to Φ_H, up to the renaming of some registers that appear in B. We have to be careful about the

following: each creation of a new position along B could lead to the creation of a new register later in B, which in turn could create a new renamed copy of some formula of Φ_H, which then could lead to the creation of another position. We must make sure that such a process cannot go ad infinitum. Let us first establish that the number of free registers in hypersequents of B is bounded by $4 \cdot |H| + \mathsf{len}(H)$. Because we always unify registers with $(\uparrow \vdash)$ as soon as possible in B (condition (c) of being a partial proof), and because the only way to introduce new registers is via rules $(\downarrow \vdash)$ and $(\vdash \downarrow)$ when no thaw appear on the left-hand side, we always have less free registers than positions. We have at most $\mathsf{len}(H)$ positions initially; and we must now bound the creations of positions that can effectively lead to the creation of a new register later in B (new positions can only be created by rules $(\vdash \mathsf{H})$ and $(\vdash \mathsf{G})$):

- For any $\mathsf{H}\,\varphi$ among the subformulæ of B that do not contain a free register (so among $|H|$ formulæ), a new position can only be created once without creating an immediately provable hypersequent (condition (b) from Lemma 5.7).
- For any $\mathsf{G}\,\varphi$ among the subformulæ of B that do not contain a free register, a new position can only be created once in the same cluster (because of case (ii) of condition (a)). A second position cannot be created elsewhere either without creating an immediately provable hypersequent (condition (a) from Lemma 5.7).
- For any $\downarrow_r \mathsf{H}_{\neq r}\,\varphi \in \Phi_H$, many formulæ of the form $\mathsf{H}_{\neq x}\,\varphi$ could appear along B (as many as free registers), and they could all lead to the creation of a new position. However, only 2 such positions can be created along B (each time for a different x) without creating an immediately provable hypersequent.
- Similarly, for any $\downarrow_r \mathsf{G}_{\neq r}\,\varphi \in \Phi_H$, many formulæ of the form $\mathsf{G}_{\neq x}\,\varphi$ could appear along B. Let us prove that a new position can only be created at most 4 times by such formulæ (each time for a different x) without creating an immediately provable hypersequent (the worst case being two different clusters, both containing two such sequents). First of all, we cannot create 3 such positions with $i \prec j \prec k$ without creating an immediately provable hypersequent, so the worst case indeed involve at most two clusters. Moreover, we cannot create more than two such positions in the same cluster. Let us look at the evolution of such a cluster along B. A first position i is created for a formula $\mathsf{G}_{\neq x}\,\varphi$, and a second position $j \sim i$ could be created later for a formula $\mathsf{G}_{\neq y}\,\varphi$ (with $y \neq x$) only if the ith position of the current hypersequent has $\uparrow_y$ on its left-hand side. But then, a third creation of such a position (for a formula $\mathsf{G}_{\neq z}\,\varphi$, with $z \neq y$) will be prevented since j is an instance of (iii) (the jth position of the current hypersequent cannot contain $\uparrow_z$ on its left-hand side, since it already contains $\uparrow_y$ and we always unify registers as soon as possible).
- One more (overall) position could be created, leading to an immediately provable hypersequent. In such a case, a new register will not be created since the hypersequent as to be proved immediately as sketched in the proof of Lemma 5.7.
- Because of condition (c), whenever a new cell is created by a formula $\mathsf{H}_{=x}\,\varphi$ or $\mathsf{G}_{=x}\,\varphi$, $\uparrow_x$ will appear on its left-hand side from the next step in B. Hence, such new cells will not lead to new registers, and the number of registers is bounded by $4 \cdot |H| + \mathsf{len}(H)$. Moreover, at most one such cell can be created for every φ (among $|H|$ formulæ), and every register appearing in B, i.e. $(4 \cdot |H| + \mathsf{len}(H)) \cdot |H|$ in total.

This also proves that the number of positions of hypersequents in B is at most $(4 \cdot |H| + \mathsf{len}(H)) \cdot |H| + \mathsf{len}(H) + 1$. Now, any other rule application adds some subformulæ among

$(4 \cdot |H| + \mathsf{len}(H)) \cdot |H|$ to the left or to the right of the turnstile at a position among $(4 \cdot |H| + \mathsf{len}(H)) \cdot |H| + \mathsf{len}(H) + 1$, hence with $2 \cdot (4 \cdot |H| + \mathsf{len}(H)) \cdot |H| \cdot ((4 \cdot |H| + \mathsf{len}(H)) \cdot |H| + \mathsf{len}(H) + 1)$ choices. Thus B is of length at most $2 \cdot |H| \cdot (4 \cdot |H| + \mathsf{len}(H)) \cdot ((4 \cdot |H| + \mathsf{len}(H)) \cdot |H| + \mathsf{len}(H) + 1)$. □

Example 5.11. If we did not follow our strategy, a bad case such as described at the beginning of the previous proof could happen on the hypersequent $H = \vdash \; ; \; \varphi \vdash$ with $\varphi = \mathsf{H}\,(\neg\downarrow_r \mathsf{H}_{=r} \bot)$. In practice, φ can send the subformula $\downarrow_r \mathsf{H}_{=r} \bot$ on the right-hand side of any past position by using ($\mathsf{H} \vdash$) and then handling the negation. A proof of H will start as follows:

$$\dfrac{\dfrac{\dfrac{\mathsf{H}_{=x} \bot \vdash \uparrow_x \supset \bot \; ; \varphi, \uparrow_x \vdash \mathsf{H}_{=x} \bot, \downarrow_r \mathsf{H}_{=r} \bot \; ; \varphi \vdash}{\varphi, \uparrow_x \vdash \mathsf{H}_{=x} \bot, \downarrow_r \mathsf{H}_{=r} \bot \; ; \varphi \vdash}\,(\vdash \mathsf{H})}{\varphi \vdash \downarrow_r \mathsf{H}_{=r} \bot \; ; \varphi \vdash}\,(\vdash \downarrow)}{\vdash \; ; \varphi \vdash}$$

Our strategy would now force to handle the formula $\uparrow_x \supset \bot$. If we do not respect that, and instead send $\downarrow_r \mathsf{H}_{=r} \bot$ on the right-hand side of the leftmost position and apply ($\vdash \downarrow$) again, a new register y would be created, along with the formula $\mathsf{H}_{=y} \bot$, which in turn will create a new position more in the past when applying ($\vdash \mathsf{H}$). If we never deal with a formula of the form $\uparrow_x \supset \bot$, this process could go on ad infinitum, alternating between creating a new register and creating a new position. However, if we respect our strategy, the proof search will reach an immediately provable hypersequent after creating a fourth position. It is not surprising, since we can prove that a counter-model of H should be such that every datum appearing in the past does so infinitely many times, which is impossible as our models are well-founded.

We now conclude that $\mathbf{HK}^{\mathbf{d}}_{\mathbf{t}}\mathbf{L}_{\boldsymbol{\ell}}\mathbf{.3}$ is complete for $\mathbf{K}^{\mathbf{d}}_{\mathbf{t}}\mathbf{L}_{\ell}\mathbf{.3}$, and also enjoys proof search of optimal complexity.

Theorem 5.12 (completeness). *Every valid hypersequent H has a proof in* $\mathbf{HK}^{\mathbf{d}}_{\mathbf{t}}\mathbf{L}_{\boldsymbol{\ell}}\mathbf{.3}$.

Proof. Assume that H is not provable. Consider a partial proof $\mathcal{P}$ of H that cannot be expanded any more: its unjustified leaves are failure hypersequents. Such a partial proof exists by Lemma 5.10. Any unjustified leaf of that partial proof has a counter-model by Proposition 5.9, and by invertibility shown in Proposition 5.5 it is also a counter-model of H. □

Proposition 5.13. *Proof search in* $\mathbf{HK}^{\mathbf{d}}_{\mathbf{t}}\mathbf{L}_{\boldsymbol{\ell}}\mathbf{.3}$ *is in* coNP.

Proof. Proof search can still be implemented in an alternating Turing machine maintaining the current hypersequent on its tape, where existential states choose which rule to apply (and how) and universal states choose a premise of the rule. By Lemma 5.10, the computation branches are of length bounded by a polynomial. By Proposition 5.5, the non-deterministic choices in existential states can be replaced by arbitrary deterministic choices; and by Lemma 5.1, the choice of a fresh x by any application of ($\downarrow \vdash$) or ($\vdash \downarrow$) does not matter (e.g., x can be taken as the next unused integer), thus the resulting Turing machine has only universal states, hence is in coNP. □

5.5. Restricted Logic on Given Ordinals

We have designed a proof system that is sound and complete for $\mathbf{K}^{\mathbf{d}}_{\mathbf{t}}\mathbf{L}_{\ell}\mathbf{.3}$, and enjoys optimal complexity proof search. We now observe that the logic can only distinguish ordinals up to ω^2, as the underlying data-free logic (Section 4.5).

Proposition 5.14 (small model property). *If a hypersequent H has a counter-model, then it has a counter-model of order type $\alpha < \omega \cdot ((4|H| + \mathsf{len}(H))|H| + \mathsf{len}(H) + 1)$.*

Proof. This is a corollary of Theorem 5.12. By the proof of Lemma 5.10, the hypersequents in a failure hypersequent—which are not immediately provable—have at most $(4|H| + \mathsf{len}(H))|H| + \mathsf{len}(H) + 1$ non-empty sequents. The counter-model extracted in Proposition 5.9 from a failure hypersequent H' is over $o(H') < \omega \cdot ((4|H| + \mathsf{len}(H))|H| + \mathsf{len}(H) + 1)$. A counter-model for H is then obtained by Proposition 5.5, with a different embedding but the same structure. □

In particular, for a formula φ, the hypersequent $H = \vdash \varphi$ has $|H| = |\varphi|$ and $\mathsf{len}(H) = 1$, hence the $\omega \cdot (4 \cdot |\varphi|^2 + |\varphi| + 2)$ bound announced in the introduction.

Finally, as in Chapter 4, a free outcome of our approach is that this system can still easily be enriched—exactly as in Section 4.5—to obtain decision procedures not only for tense data logic over arbitrary ordinals, but also to capture validity over ordinals below some $\omega \cdot k + m$, or to reason over a specific ordinal of this form.

5.6. First-Order Logic with Two Variables

In this section, we show that $\mathbf{K}_\mathbf{t}^\mathbf{d}\mathbf{L}_\ell.\mathbf{3}$ is exactly as expressive as the two-variable fragment of first-order logic over data ordinals from Bojańczyk et al. [2011].

5.6.1. Syntax and Semantics. We consider first-order formulæ with two variables x and y over the signature $(=, \sim, <, (p)_{p\in\Phi})$ where $=$, $<$ and $\sim$ are binary relational symbols and each p is a unary relational symbol:

$$\psi ::= z = z' \mid z < z' \mid z \sim z' \mid p(z) \mid \bot \mid \psi \supset \psi \mid \forall z.\psi \qquad \text{(first-order formulæ)}$$

where z, z' range over $\{x, y\}$ and p over Φ. We call this logic $\mathrm{FO}^2(\sim, <)$.

We interpret our formulæ over structures $\mathfrak{M} = ((\alpha, \delta), V)$ where $=$ is interpreted as the equality over α, $<$ as the canonical strict total ordering of α, $\sim$ as the equality with respect to δ, and each p as $V(p)$ for the valuation $V : \Phi \to 2^W$.

That is, we say that $\mathfrak{M}$ satisfies ψ under an assignment $\sigma : \{x, y\} \to \alpha$, written $\mathfrak{M}, \sigma \models \psi$, in the following inductive cases:

$$\begin{array}{ll}
\mathfrak{M}, \sigma \not\models \bot & \\
\mathfrak{M}, \sigma \models z = z' & \text{if } \sigma(z) = \sigma(z') \\
\mathfrak{M}, \sigma \models z < z' & \text{if } \sigma(z) < \sigma(z') \\
\mathfrak{M}, \sigma \models z \sim z' & \text{if } \delta(\sigma(z)) = \delta(\sigma(z')) \\
\mathfrak{M}, \sigma \models p(z) & \text{if } \sigma(z) \in V(p) \\
\mathfrak{M}, \sigma \models \psi \supset \psi' & \text{if } \mathfrak{M}, \sigma \models \psi \text{ implies } \mathfrak{M}, \sigma \models \psi' \\
\mathfrak{M}, \sigma \models \exists z.\psi & \text{if } \exists w \in W,\ \mathfrak{M}, \sigma[w/z] \models \psi
\end{array}$$

where $\sigma[w/z]$ is the updated assignment mapping z to w and the remaining variable $z' \in \{x, y\} \setminus \{z\}$ to $\sigma(z')$.

5.6.2. Equivalence with $\mathbf{K_t^d L_\ell}.\mathbf{3}$. Given an $\mathrm{FO}^2(\sim, <)$ formula $\psi(z)$ with one free variable z, we show how to construct a $\mathbf{K_t^d L_\ell}.\mathbf{3}$ formula φ such that, for all data ordinals $\mathfrak{M} = ((\alpha, \delta), V)$, $\mathfrak{M}, [w/z] \models \psi$ if and only if $\mathfrak{M}, w \models \varphi$, where $[w/z]$ is the variable assignment mapping z to w.

THEOREM 5.15. *Every* $\mathrm{FO}^2(\sim, <)$ *formula* $\varphi(x)$ *can be converted to an equivalent* $\mathbf{K_t^d L_\ell}.\mathbf{3}$ *formula* φ' *with* $|\varphi'| \in 2^{poly(|\varphi|)}$.

PROOF. The proof works the same as in Etessami et al. [2002] for $\mathrm{FO}^2(<)$, which is recalled in Fact 2.7. We sketch how to adapt it for data ordinals.

After putting φ in Scott normal form—as in Etessami et al. [2002]—we construct a translation by structural induction. At this point, assuming by induction hypothesis that ψ' is the translation of some formula $\psi(x)$, we need to provide translation to a formula of the form $\exists y(\tau(x, y) \wedge \psi(y))$, where τ is what we call a *data order type*, and expresses which relations hold between x and y (in Etessami et al. [2002], $\tau(x, y)$ is called an order type and expresses which order relation holds between x and y). We consider 9 mutually exclusive cases of such $\tau(x, y)$ in the following table, where $\tau\langle\psi\rangle$ denotes the translation of $\exists y(\tau(x, y) \wedge \psi(y))$:

$\tau(x, y)$	$\tau\langle\psi\rangle$
$x = y$	ψ'
$x \sim y$	$\psi' \vee \downarrow_r \mathsf{F}_{=r}\, \psi' \vee \downarrow_r \mathsf{P}_{=r}\, \psi'$
$\neg(x \sim y)$	$\downarrow_r \mathsf{F}_{\neq r}\, \psi' \vee \downarrow_r \mathsf{P}_{\neq r}\, \psi'$
$x < y$	$\mathsf{F}\, \psi'$
$x < y \wedge x \sim y$	$\downarrow_r \mathsf{F}_{=r}\, \psi'$
$x < y \wedge \neg(x \sim y)$	$\downarrow_r \mathsf{F}_{\neq r}\, \psi'$
$y < x$	$\mathsf{P}\, \psi'$
$y < x \wedge x \sim y$	$\downarrow_r \mathsf{P}_{=r}\, \psi'$
$y < x \wedge \neg(x \sim y)$	$\downarrow_r \mathsf{P}_{\neq r}\, \psi'$

Any other $\tau(x, y)$ can be reduced to either these cases, or trivially to $\bot$ or $\top$. □

Conversely, $\mathbf{K_t^d L_\ell}.\mathbf{3}$ formulæ can be easily translated into $\mathrm{FO}^2(\sim, <)$ formulæ. Hence, $\mathbf{K_t^d L_\ell}.\mathbf{3}$ and $\mathrm{FO}^2(\sim, <)$ are equally expressive.

Thus, as in the previous chapters, our results yield an optimal NEXP upper bound for the satisfiability of $\mathrm{FO}^2(\sim, <)$.

5.7. Related Work and Conclusion

We have investigated $\mathbf{K_t^\downarrow L_\ell}.\mathbf{3}$—the freeze tense logic over ordinals—and proposed a decidable fragment, namely $\mathbf{K_t^d L_\ell}.\mathbf{3}$, for which we designed a sound and complete proof system. This fragment is equally expressive as $\mathrm{FO}^2(\sim, <)$ from Bojańczyk et al. [2011]. It is also a fragment of the Logic of Repeating Values (LRV) from Demri et al. [2016], even though formulæ of the form $\downarrow_r \mathsf{H}_{=r}\, \varphi$ and $\downarrow_r \mathsf{H}_{\neq r}\, \varphi$ can only be encoded in the enriched version of their logic with past modalities (PLRV), for which the satisfiability problem is equivalent to the problem of reachability in VASS, which is TOWER-hard [Czerwiński et al., 2019], and with an ACKERMANN complexity upper bound [Leroux and Schmitz, 2019]. However, the satisfiability problem of $\mathbf{K_t^d L_\ell}.\mathbf{3}$ has a smaller complexity (NP), as established by our proof system.

Thanks to Indrzejczak's ordered hypersequents[Indrzejczak, 2016], enriched with clusters and annotations as in Chapters 3 and 4, our system enjoys optimal coNP proof search, allows

to derive small model properties, and can be extended into a proof system for variants of the logic over bounded or fixed data ordinals. The main contribution of this chapter is the ability to maintain the small branch property of the calculus with the addition of data registers.

Part 2

XPath in the Real World

CHAPTER 6

Real-World XPath

Contents

6.1. Introduction

The satisfiability of XPath queries has been widely studied, as this abstract question actually allows to answer several questions on the reliability and performance of a query. Unfortunately, the satisfiability problem is in general undecidable, and most of the decidable fragments are intractable. For instance, the pure navigational fragment CoreXPath 1.0 is EXP-complete already when only using the `child` axis [Afanasiev et al., 2005; Benedikt et al., 2008; Neven and Schwentick, 2006]. Thus, most of the literature on the topic is of a theoretical nature [e.g. Abriola et al., 2017b; Baelde et al., 2016; Bojańczyk et al., 2009; Figueira, 2012a,b, 2013; Figueira and Segoufin, 2017; Geerts and Fan, 2005; Jurdziński and Lazić, 2011], and focuses on decidability and complexity questions in variants of CoreXPath 1.0 that allow either limited forms of data joins or restricted navigation with full data joins.

However, handling data joins may not necessarily imply much more coverage in practice, as they are not the only source of difficulty in XPath: many real-life XPath queries perform calls to a standard library of functions [xfu, 2014]—including arithmetic and string-manipulating functions—that also lead to undecidable satisfiability.

Also, XPath 1.0 dates back to 1999; the more recent versions 2.0 and 3.0 feature path intersections, `for` loops, etc. [ten Cate and Lutz, 2009]. As it evolves in pace with XQuery, XPath

includes more and more general programming constructs, and is arguably not just a domain-specific language for path navigation—quite tellingly, we shall see that, among the real-world queries we gathered, only half of them used navigation.

In this part, we evaluate the practical coverage of XPath fragments proposed in the theoretical literature by measuring how many real-world queries are captured by these fragments. The first step to this end, presented in Chapter 6, is the compilation of a benchmark of 21,141 real-world XPath queries, which are extracted from XSLT or XQuery open-source projects. As described in Section 6.3, each XPath query is parsed by our benchmark and an XML syntax tree representation is output in XQueryX [xqu, 2014b]. The tools and the resulting benchmark are available from Baelde et al. [2019b,c] under open source licenses.

We then try to test the queries in our benchmark against the syntax allowed in theoretical works on XPath satisfiability, namely by writing Relax NG specifications for PositiveXPath [Geerts and Fan, 2005; Hidders, 2004], CoreXPath 1.0 [Gottlob and Koch, 2002], CoreXPath 2.0 [ten Cate and Lutz, 2009], fragments of DataXPath [Figueira, 2012a,b; Figueira and Segoufin, 2017], and fragments of XPath that can be interpreted in EMSO^2 [Bojańczyk et al., 2009] or using non-mixing MSO constraints [Czerwinski et al., 2017] (see Chapter 7).

Naturally, the syntax defined in these works is simplified and was never meant to be used directly against concrete XPath inputs, while we need a concrete syntax for each one of these fragments in order to implement it in Relax NG, as our benchmark contains real XPath queries. As a result, a naive implementation of the syntactic fragments from the literature does not lead to a good coverage (see Section 7.3). Thus, we investigate in Chapter 8 which XPath features can be 'reasonably' handled in these fragments without losing decidable satisfiability nor hampering its complexity. To that end, in Section 6.2 we provide an agreeable semantics for a substantial subset of XPath 3.0.

In Chapter 8, we propose six extensions of the original fragments and evaluate their coverage on the benchmark. These extensions are root navigation, free variables, data tests against constants, positive data joins, and restricted calls to the functions `last()` and `id()`. Just as interestingly, we exhibit several cases where these extensions *cannot* be handled.

We analyse our experimental results in Section 8.5, notably concluding that higher coverage tends to be obtained through basic extensions rather than by using complex academic fragments. We also identify increased function support as a promising direction for improved practical satisfiability checking, with an especially high potential for XPath queries from XSLT sources. We conclude in Section 8.6.

6.2. XPath 3.0

The XPath 3.0 specification is arguably too complex to be reasoned about directly. We work instead with a well-defined sub-language, designed to capture accurately the constructions we witnessed in the benchmark. In order to be compatible with the semantics in the XPath literature, we provide a semantics on data trees, but in Section 6.2.6 we show how to capture the actual XPath semantics on XML documents.

6.2.1. Data Trees. Our models are an abstraction of XML DOM trees called *data trees*, which are finite trees where each node carries both a *label* from a finite alphabet Σ and a *datum* from an infinite countable domain $\mathbb{D}$ equipped with an order $<$.

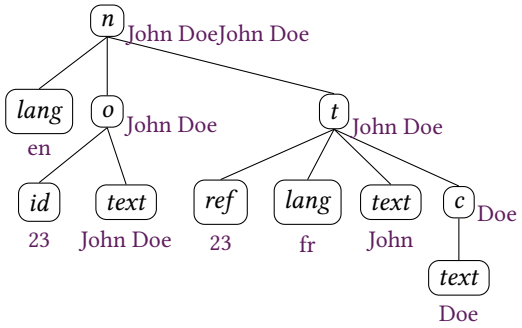

FIGURE 6.1. A data tree; labels from Σ are shown in the nodes, data from $\mathbb{D}$ in violet next to them.

Formally, a data tree is a finite rooted ordered unranked tree with labels in $\Sigma \times \mathbb{D}$: it is a pair $t = (\ell, \delta)$ of functions $\ell\colon N \to \Sigma$, $\delta\colon N \to \mathbb{D}$ with a common non-empty finite set of nodes $N \subseteq \mathbb{N}^*$ as domain; N must be prefix-closed (if $p \cdot i \in N$ for some $p \in \mathbb{N}^*$ and $i \in \mathbb{N}$, then $p \in N$) and predecessor-closed (if $p \cdot (i+1) \in N$ for some $p \in \mathbb{N}^*$ and $i \in \mathbb{N}$, then $p \cdot i \in N$); in particular it contains a root node ε. Nodes in N, being finite sequences of natural numbers, are totally ordered by the lexicographic ordering, which is known in this context as the *document order* and denoted by $\ll$. Figure 6.1 displays an example of a data tree. When working with first-order or monadic second-order logic, a data tree is seen as a finite relational structure $(N, \downarrow, \rightarrow, \sim, (P_a)_{a\in\Sigma}, (P_d)_{d\in\mathbb{D}})$ where

$$\begin{aligned} \rightarrow &= \{(p\cdot i, p\cdot(i+1)) \in N^2 \mid p \in \mathbb{N}^*, i \in \mathbb{N}\}\ , \\ \downarrow &= \{(p, p\cdot i) \in N^2 \mid p{\in}\mathbb{N}^*, i{\in}\mathbb{N}\}\ , & P_a &= \{p \in N \mid \ell(p){=}a\}, \\ \sim &= \{(p, p') \in N^2 \mid \delta(p) = \delta(p')\}\ , & P_d &= \{p \in N \mid \delta(p){=}d\} \end{aligned}$$

denote respectively the child relation, the next-sibling relation, data equivalence, and the labelling and data predicates.

6.2.2. Syntax. While our implementation works with concrete syntax (see e.g. the example in Example 6.4), for the sake of readability we use an abstract syntax throughout this part. It is nevertheless fully compatible with XPath 3.0: all the examples in this part are written in actual XPath.

Let $\mathcal{X}$ be a countable infinite set of variables, and $\mathcal{F}$ be a ranked alphabet of function names; we denote by $\mathcal{F}_n$ its subset of symbols with arity n. As usual, our language has multiple sorts: *axes* denote directions in the data tree, with abstract syntax

$$\begin{aligned} \alpha ::= {} & \texttt{self} \mid \texttt{child} \mid \texttt{descendant} \mid \texttt{following-sibling} \\ & \mid \texttt{parent} \mid \texttt{ancestor} \mid \texttt{preceding-sibling} \end{aligned}$$

path expressions describe binary relations between the nodes in a data tree, with abstract syntax

$$\begin{aligned} \pi ::= {} & \alpha{::}* \mid /\pi \mid \pi/\pi \mid \pi[\varphi] \mid \pi \ \texttt{union}\ \pi \mid f(\pi_1, \ldots, \pi_n) \\ & \mid \$\texttt{x} \mid \texttt{let}\ \$\texttt{x} := \pi\ \texttt{return}\ \pi \mid \texttt{for}\ \$\texttt{x}\ \texttt{in}\ \pi\ \texttt{return}\ \pi \end{aligned}$$

$$\begin{aligned}
[\![\texttt{self}]\!]_{\mathsf{A}} &= \{(p,p) \mid p \in N^2\} \\
[\![\texttt{child}]\!]_{\mathsf{A}} &= \downarrow & [\![\texttt{parent}]\!]_{\mathsf{A}} &= \downarrow^{-1} \\
[\![\texttt{following-sibling}]\!]_{\mathsf{A}} &= \rightarrow^{+} & [\![\texttt{ancestor}]\!]_{\mathsf{A}} &= (\downarrow^{-1})^{+} \\
[\![\texttt{preceding-sibling}]\!]_{\mathsf{A}} &= (\rightarrow^{-1})^{+} & [\![\texttt{descendant}]\!]_{\mathsf{A}} &= \downarrow^{+}
\end{aligned}$$

FIGURE 6.2. The semantics of XPath axes.

where $x ranges over $\mathcal{X}$, n over $\mathbb{N}$, and f over $\mathcal{F}_n$, while *node expressions* describe sets of nodes, with abstract syntax

$$\varphi ::= \pi \mid a \mid \texttt{false}() \mid \texttt{not}(\varphi) \mid \varphi \texttt{ or } \varphi \mid \pi \texttt{ is } \pi \mid \pi \triangle \pi \mid \pi \triangle^{+} d$$

where a ranges over Σ, d over $\mathbb{D}$, $\triangle$ over $\{\texttt{eq}, \texttt{ne}\}$ and $\triangle^{+}$ over $\{\texttt{eq}, \texttt{ne}, \texttt{le}, \texttt{lt}, \texttt{ge}, \texttt{gt}\}$. Note that only eq and ne are allowed when comparing paths, while the ordered structure of $\mathbb{D}$ is only available when comparing a path and a data constant: none of the fragments we consider is known to allow the richer path comparisons. We provide a detailed breakdown of how many queries from our benchmark use each one of these syntactic constructs in Section 6.3.4.

6.2.3. Data Tree Semantics. For a fixed data tree of domain N, we give in Figures 6.2 and 6.3 the semantics of axes $[\![\alpha]\!]_{\mathsf{A}}$, path and node expressions $[\![\pi]\!]^{\nu}_{\mathsf{P}}$ and $[\![\varphi]\!]^{\nu}_{\mathsf{N}}$. The semantics is relative to a current variable valuation $\nu\colon \mathcal{X} \to 2^N$. The semantics of node expressions are sets of nodes of N, while those of axes and path expression are sets of pairs of nodes.

In order to interpret functions, each function symbol f from $\mathcal{F}_n$ comes with a semantics $[\![f]\!]_{\mathcal{F}}\colon (2^N)^n \to 2^N$. For instance, false() and not() are technically XPath functions, with semantics $[\![\texttt{false}]\!]_{\mathcal{F}} = \emptyset$ and $[\![\texttt{not}]\!]_{\mathcal{F}}(S) = N \setminus S$ for all $S \subseteq N$. Likewise, we interpret each comparison operator $\triangle^{+}$ as a data relation $\overset{\triangle^{+}}{\sim} \subseteq \mathbb{D} \times \mathbb{D}$: $\overset{\texttt{eq}}{\sim}$ is the equality $=$ over $\mathbb{D}$, and $\overset{\texttt{ne}}{\sim}$ the disequality $\neq$, $\overset{\texttt{lt}}{\sim}$ the strict order $<$, etc. For binary relations R, R', we employ relational compositions $R \fatsemi R' = \{(p,p'') \mid \exists p'.(p,p') \in R \wedge (p',p'') \in R'\}$, transitive closures R^{+}, converses $R^{-1} = \{(p',p) \mid (p,p') \in R\}$, and images $R(p) = \{p' \mid (p,p') \in R\}$.

Beware that the semantics of a path expression π changes when seen as a node expression. In particular, a variable $x is a *path expression*, and when seen as a node formula, $[\![\$x]\!]^{\nu}_{\mathsf{N}} = N$ unless $\nu(\$x) = \emptyset$. XPath provides two quantifiers: for expressions bind singleton node sets, while let expressions bind node sets.

EXAMPLE 6.1. Evaluating the query

$$\texttt{for } \$x \texttt{ in child::}*[o \texttt{ or } t] \texttt{ return } \$x[\texttt{self::}* \texttt{ eq } \$x/\texttt{child::}*]$$

at the root of the data tree in Figure 6.1 binds $x to each of the two children nodes labelled o or t in succession, and returns the o node. Evaluating

$$\texttt{let } \$x := \texttt{child::}*[o \texttt{ or } t] \texttt{ return } \$x[\texttt{self::}* \texttt{ eq } \$x/\texttt{child::}*]$$

at the root binds $x to the set containing both o and t, and returns both of them.

Our semantics is in line with the ones found in the literature. However, we note that it slightly differs from the actual XPath semantics. In particular, we only account for pure functions acting on paths, while functions from XPath's large standard library [xfu, 2014] may be polymorphic and have side-effects—two features that are anyway out of the reach of

$$
\begin{aligned}
[\![\alpha :: *]\!]^\nu_{\mathsf{P}} &= [\![\alpha]\!]_{\mathsf{A}} \\
[\![/\pi]\!]^\nu_{\mathsf{P}} &= N \times [\![\pi]\!]^\nu_{\mathsf{P}}(\varepsilon) \\
[\![\pi/\pi']\!]^\nu_{\mathsf{P}} &= [\![\pi]\!]^\nu_{\mathsf{P}} \fatsemi [\![\pi']\!]^\nu_{\mathsf{P}} \\
[\![\pi[\varphi]]\!]^\nu_{\mathsf{P}} &= [\![\pi]\!]^\nu_{\mathsf{P}} \cap (N \times [\![\varphi]\!]^\nu_{\mathsf{N}}) \\
[\![\pi \texttt{ union } \pi']\!]^\nu_{\mathsf{P}} &= [\![\pi]\!]^\nu_{\mathsf{P}} \cup [\![\pi']\!]^\nu_{\mathsf{P}} \\
[\![f(\pi_1, \ldots, \pi_n)]\!]^\nu_{\mathsf{P}} &= \{(p, [\![f]\!]_{\mathcal{F}}([\![\pi_1]\!]^\nu_{\mathsf{P}}(p), \ldots, [\![\pi_n]\!]^\nu_{\mathsf{P}}(p))) \mid p \in N\} \\
[\![\$\mathrm{x}]\!]^\nu_{\mathsf{P}} &= N \times \nu(\$\mathrm{x}) \\
[\![\texttt{let } \$\mathrm{x} := \pi \texttt{ return } \pi']\!]^\nu_{\mathsf{P}} &= \{(p, [\![\pi']\!]^{\nu[\$\mathrm{x} \mapsto [\![\pi]\!]^\nu_{\mathsf{P}}(p)]}_{\mathsf{P}}(p)) \mid p \in N\} \\
[\![\texttt{for } \$\mathrm{x} \texttt{ in } \pi \texttt{ return } \pi']\!]^\nu_{\mathsf{P}} &= \{(p, p'') \in [\![\pi']\!]^{\nu[\$\mathrm{x} \mapsto \{p'\}]}_{\mathsf{P}} \mid p' \in [\![\pi]\!]^\nu_{\mathsf{P}}(p)\}
\end{aligned}
$$

$$
\begin{aligned}
[\![\pi]\!]^\nu_{\mathsf{N}} &= \{p \in N \mid \exists p' \in [\![\pi]\!]^\nu_{\mathsf{P}}(p)\} \\
[\![a]\!]^\nu_{\mathsf{N}} &= P_a \\
[\![\texttt{false}()]\!]^\nu_{\mathsf{N}} &= \emptyset \\
[\![\texttt{not}(\varphi)]\!]^\nu_{\mathsf{N}} &= N \setminus [\![\varphi]\!]^\nu_{\mathsf{N}} \\
[\![\varphi \texttt{ or } \varphi']\!]^\nu_{\mathsf{N}} &= [\![\varphi]\!]^\nu_{\mathsf{N}} \cup [\![\varphi']\!]^\nu_{\mathsf{N}} \\
[\![\pi \texttt{ is } \pi']\!]^\nu_{\mathsf{N}} &= \{p \in N \mid \exists p' \,.\, \{p'\} = [\![\pi]\!]^\nu_{\mathsf{P}}(p) = [\![\pi']\!]^\nu_{\mathsf{P}}(p)\} \\
[\![\pi \mathbin{\triangle^+} d]\!]^\nu_{\mathsf{N}} &= \{p \in N \mid \exists p' \in [\![\pi]\!]^\nu_{\mathsf{P}}(p) \,.\, \delta(p') \overset{\triangle^+}{\sim} d\} \\
[\![\pi \mathbin{\triangle} \pi']\!]^\nu_{\mathsf{N}} &= \{p \in N \mid \exists p', p'' \,.\, p' \in [\![\pi]\!]^\nu_{\mathsf{P}}(p) \\
&\qquad\qquad \wedge\, p'' \in [\![\pi']\!]^\nu_{\mathsf{P}}(p) \wedge \delta(p') \overset{\triangle}{\sim} \delta(p'')\}
\end{aligned}
$$

FIGURE 6.3. The semantics of XPath path expressions and node expressions.

the current decidable fragments. Also, variables in XPath are bound to ordered collections of nodes and data values, while we only consider sets of nodes. This simpler semantics is not restrictive for our purposes, as discussed further in Section 6.2.6.

6.2.4. The Satisfiability Problem. In this part, we focus on the *satisfiability problem*: given a node expression φ, does there exist a data tree t such that $t \models \varphi$? As path expressions are also node expressions, this also captures the satisfiability of path expressions. In presence of a DTD, the data tree t should additionally belong to the DTD's language.

REMARK 6.2. A related problem is *query containment*: for node expressions φ and φ', we say that φ is *contained* in φ' if, for all data trees and all variable valuations ν, $[\![\varphi]\!]^\nu_{\mathsf{N}} \subseteq [\![\varphi']\!]^\nu_{\mathsf{N}}$. This is equivalent to asking the unsatisfiability of φ and $\texttt{not}(\varphi')$, so it reduces to the satisfiability problem when negation is allowed—which will not always be our case. The problem of *path containment* asks the same question for path expressions π and π', and is not captured by satisfiability. Furthermore, one might also consider variants of these problems where we ask instead whether for all data trees and variable valuations ν and ν', $[\![\varphi]\!]^\nu_{\mathsf{N}} \subseteq [\![\varphi']\!]^{\nu'}_{\mathsf{N}}$, which leads to rather different complexities [Neven and Schwentick, 2006].

6.2.5. Syntactic Sugar. XPath comes with some handy syntactic sugar.

6.2.5.1. *XPath 1.0 Sugar.* Beside standard definitions like $\texttt{true}() = \texttt{not}(\texttt{false}())$ or $\varphi \texttt{ and } \varphi' = \texttt{not}(\texttt{not}(\varphi) \texttt{ or } \texttt{not}(\varphi'))$, we may use $. = \texttt{self}::*$ for referring to the current point of focus, $.. = \texttt{parent}::*$ for its parent, $\alpha::a = \alpha::*[a]$ for testing the label found after an axis step, and a single label a as a path formula for $\texttt{child}::a$.

The syntax also features more axes:

$$\begin{aligned}
\texttt{descendant-or-self}::* &= \texttt{descendant}::* \texttt{ union } \texttt{self}::* \\
\texttt{ancestor-or-self}::* &= \texttt{ancestor}::* \texttt{ union } \texttt{self}::* \\
\texttt{following}::* &= \texttt{ancestor-or-self}::*/\texttt{following-sibling}::*/ \\
&\qquad \texttt{descendant-or-self}::* \\
\texttt{preceding}::* &= \texttt{ancestor-or-self}::*/\texttt{preceding-sibling}::*/ \\
&\qquad \texttt{descendant-or-self}::*.
\end{aligned}$$

The shorthand $\pi//\pi'$ stands for $\pi/\texttt{descendant-or-self}::*/\pi'$, and $//\pi$ is defined similarly by $/\texttt{descendant-or-self}::*/\pi$. Conditional node expressions, while only available in XPath 2.0 and later, are also easily handled: $\texttt{if } (\varphi) \texttt{ then } \varphi' \texttt{ else } \varphi'' = (\varphi \texttt{ and } \varphi') \texttt{ or } (\texttt{not}(\varphi) \texttt{ and } \varphi'')$.

6.2.5.2. *XPath 2.0 Sugar.* As shown by ten Cate and Lutz [2009], path intersection and complementation as introduced in XPath 2.0 can be expressed using `for` loops: $\pi \texttt{ intersect } \pi'$ is defined by

$$\texttt{for \$x in } \pi \texttt{ return for \$y in } \pi' \texttt{ return \$x[\$x is \$y]} \tag{6.1}$$

and $\pi \texttt{ except } \pi'$ by

$$\texttt{for \$x in } \pi \texttt{ return .[not(}\pi'\texttt{[. is \$x])]/\$x} \tag{6.2}$$

Similarly, node quantification $\texttt{some \$x in } \pi \texttt{ satisfies } \varphi$ can be expressed using

$$\texttt{for \$x in } \pi \texttt{ return .[}\varphi\texttt{]} \tag{6.3}$$

and $\texttt{every \$x in } \pi \texttt{ satisfies } \varphi$ is defined dually. By first defining the non-standard $\mathit{future}(\varphi) = (\texttt{following}::* \texttt{ union } \texttt{descendant}::*)[\varphi]$ and $\mathit{singleton}(\pi) = \pi \texttt{ and not}(\pi \texttt{ intersect } \pi/\mathit{future}(\texttt{true}()))$, then we can also express standard *node comparisons* $\pi \ll \pi'$ with

$$\mathit{singleton}(\pi) \texttt{ and } \mathit{singleton}(\pi') \texttt{ and } \pi' \texttt{ intersect } (\pi/\mathit{future}(\texttt{true}())) \tag{6.4}$$

In XPath, a path expression can select the last (according to the document order) node among those selected by a path π. This often appears as a predicate $\pi[\texttt{last}()]$ or $\pi[\texttt{position}() = \texttt{last}()]$ that calls the nullary $\texttt{last}()$ function [xfu, 2014]. However, this syntax cannot be handled with our simplified semantics—and is a bit problematic—, thus we shall only consider the one-argument version of $\texttt{last}()$ [xfu, 2014], with semantics $[\![\texttt{last}]\!]_{\mathcal{F}}(S) = \max_{\ll}(S)$ for any $S \subseteq N$. Then $\texttt{last}(\pi)$ can be expressed in XPath 2.0 by

$$\pi \texttt{ except } (\pi/\texttt{ancestor}::* \texttt{ union } \texttt{preceding}::*) \tag{6.5}$$

Example 6.3. In the data tree of Figure 6.1, when evaluated at the c node, the path

$$\texttt{last}(\texttt{ancestor-or-self}::*/\texttt{child}::\mathit{lang})$$

returns the *lang* node with data value 'fr.'

We will discuss the $\texttt{last}()$ function further in Section 8.3.2.

6.2.6. XML Semantics. The XPath data model [xpd, 2014] specifies that nodes can fall into several categories, with multiple types and accessors. In data trees, there are only nodes, and two accessors ℓ and δ. Nevertheless, a large part of the XPath data model can be handled.

6.2.6.1. *Data Tree of an XML Document.* Elements, attributes, text nodes, and comments can all be encoded using distinguished labels: we let $\Sigma = E \uplus A \uplus \{\mathit{text}, \mathit{comment}\}$ where E is the set of element labels and A of attribute labels.

We see all the values as belonging to $\mathbb{D}$. The data in $\mathbb{D}$ associated with attribute nodes, text nodes, and comment nodes is their string value; for an element node, it is the concatenation of the string values of all its element and text children. The following XML document corresponds to the data tree of Figure 6.1:

```
<n lang="en">
  <o id="23">John Doe</o>
  <t ref="23" lang="fr">John <c>Doe</c></t>
</n>
```

6.2.6.2. *XML-Specific Syntax.* When considering XML documents rather than data trees as models, some additional features of XPath become meaningful. We enrich the syntax with the axis

$$\alpha ::= \cdots \mid \texttt{attribute}$$

and five *node tests*

$$\tau ::= \texttt{attribute}() \mid \texttt{comment}() \mid \texttt{element}() \mid \texttt{text}()$$
$$\varphi ::= \cdots \mid \tau$$
$$\pi ::= \cdots \mid \alpha{::}\texttt{node}()$$

meant to select the appropriate node category, and new syntactic sugar: $\alpha{::}\tau = \alpha{::}\texttt{node}()[\tau]$ and $@a = \texttt{attribute}{::}*[a]$ for $a \in A$.

6.2.6.3. *Interpretation into Data Trees.* Given an XPath node expression φ with the XML semantics of xpa [2014], we interpret it as an XPath node expression $\ulcorner\varphi\urcorner$ and $\texttt{not}(//.[\mathit{notxml}])$ that uses the data tree semantics of Section 6.2.3.

The first conjunct $\ulcorner\varphi\urcorner$ is defined by induction on φ; the case of node tests τ is straightforward:

$$\ulcorner\texttt{attribute}()\urcorner = \texttt{or}_{a\in A}\, a \qquad \ulcorner\texttt{comment}()\urcorner = \mathit{comment}$$
$$\ulcorner\texttt{element}()\urcorner = \texttt{or}_{e\in E}\, e \qquad \ulcorner\texttt{text}()\urcorner = \mathit{text}$$

The semantics of atomic steps is modified to only visit element nodes, except when using the $\texttt{attribute}$ axis or the $\texttt{node}()$ test, and to forbid horizontal axes in attribute nodes: $\ulcorner\alpha{::}\texttt{node}()\urcorner = \alpha{::}*$, while $\ulcorner\alpha{::}*\urcorner$ is defined as $\texttt{child}{::}*[\texttt{or}_{a\in A}\, a]$ if $\alpha = \texttt{attribute}$, as $\alpha{::}*[\texttt{or}_{e\in E}\, e]$ if $\alpha = \texttt{parent}$ or $\alpha = \texttt{ancestor}$, and as $.[\texttt{not}(\texttt{or}_{a\in A}\, a)]/\alpha{::}*[\texttt{or}_{e\in E}\, e]$ for all the other axes. The remaining cases of the induction are by identity homomorphism.

The second conjunct $\texttt{not}(//.[\mathit{notxml}])$ ensures that the data tree is indeed the encoding of an XML document, by forbidding the node expression *notxml* everywhere in the tree. We

define it by first ensuring that attribute, comments, and text nodes are leaves:

$$(\textit{comment}\ \texttt{or}\ \textit{text}\ \texttt{or}\ \texttt{or}_{a\in A}\, a)\ \texttt{and}\ \texttt{child::}* \tag{6.6}$$

Note that this does not enforce the XML standard of having at most one a-labelled attribute for every element. This might actually be desirable, for instance for handling set-valued attributes, like the `class` ones in HTML 5. But it can be otherwise remedied with a disjunction with (6.7):

$$\cdots\ \texttt{or}\ \texttt{or}_{a\in A}(\texttt{for \$x in child::}a\ \texttt{return}\ \texttt{for \$y in child::}a\ \texttt{return .[not(\$x is \$y)]}) \tag{6.7}$$

In case we are working with a fragment without `for` and `is`, but with data joins, (6.8) ensures instead that all the a-labelled attributes share the same data

$$\cdots\ \texttt{or}\ \texttt{or}_{a\in A}(\texttt{child::}a\ \texttt{ne child::}a) \tag{6.8}$$

We might want to ensure that identifiers are unique. To simplify matters, let us assume that all the unique identifiers use the attribute name $\texttt{id} \in A$; then we add

$$\cdots\ \texttt{or}\ \big(\texttt{id and (. eq }\textit{future}\texttt{(id))}\big) \tag{6.9}$$

Finally, we should also ensure that data values are consistent throughout the tree. Remember that the value of an element node should be the concatenation of the values of its element and text children. There is no way to do this without access to string-processing functions, so the XML semantics and data tree semantics do not quite coincide. Most of the literature accordingly restricts data joins $\pi \triangle \pi'$ and data tests $\pi \triangle^{+} d$ to paths ending with an attribute step: only $\pi/@a \triangle \pi'/@a'$ and $\pi/@a \triangle^{+} d$ are allowed in their syntax. We do not enforce this restriction in our concrete syntax specifications, but the effect is limited: there are only 381 occurrences in the benchmark of a data test $\pi \triangle^{+} d$ where π is neither a function call nor a variable and does not end with an attribute step.

6.3. A Real-World Benchmark

We explain here the technical aspects of the construction of a benchmark of 21,141 queries: the parser we developed to this end (Section 6.3.1), the sources we employed (Section 6.3.2), and the way we processed the benchmark to check whether a given query belongs to a syntactic XPath fragment (Section 6.3.3). We finish the section by mentioning the limitations of the current benchmark.

6.3.1. Parser. We have slightly modified the W3C parser for XQuery 3.0 from `https://www.w3.org/2013/01/qt-applets/`, which is (almost) a superset of XPath 3.0, so that we can also use it for XPath queries extracted from XSLT documents. This parser uses a grammar automatically extracted from the language specification, so we are confident in its results. Our implementation

(1) extracts XPath queries from XQuery files, by selecting 'maximal XPath subtrees' from the XQuery syntax tree, and
(2) outputs syntax trees in the XML format XQueryX [xqu, 2014b]; this is what we process to determine to which XPath fragments each query belongs.

Duplicate queries within each source are removed.

Example 6.4. Here is an XQuery snippet from the file `functx.xqy` of HisTEI:

```
module namespace functx = "http://www.functx.com" ;
declare function functx:id-from-element
  ( $element as element()? ) as xs:string? {
    data (($element/@*[id (.) is ..])[1])
  } ;
```

The parser identifies 'data (($element/@*[id (.) is ..])[1])' as an XPath query inside this program, and returns the information from Figure 6.4 (note the normalisation of some of the syntactic sugar, like '@*' and '..'), including the syntax tree in XQueryX format [xqu, 2014b] inside the <ast> element.

This syntax tree in XQueryX format can be validated against XML Schemas or Relax NG syntactic specifications in order to check whether it fits in some XPath fragments.

6.3.2. Sources. Three XSLT and twenty-five XQuery sources have been chosen by searching through open source GitHub projects containing XSLT or XQuery files, selecting the most popular projects from which we could extract at least 50 queries. We also added one large project not hosted on GitHub, namely DocBook XSL.

The XSLT projects aim to translate enriched text documents between different formats. The XQuery projects we include are most often libraries. The detailed composition is presented in Table 6.1, along with some coverage data discussed in Section 6.3.3. We make no formal claim about the coverage of the benchmark, as it is certainly biased by the restriction to XSLT and XQuery sources. We rather see it as a first open-source release, which could be later enriched by adding XPath queries embedded in other programming languages (e.g., Python, Perl, ECMAScript).

6.3.3. Properties of the Benchmark.

Standard Coverage. We exploit this benchmark by validating the syntax trees in XQueryX format against Relax NG [rel, 2002] specifications. Table 6.1 presents the number of queries that fall within the scope of the three major revisions of the XPath standard (queries are unique as strings, per source). We can observe that the queries extracted from XSLT files are nearly all XPath 1.0 queries, which contrasts with queries extracted from XQuery sources, which rely more often on advanced XPath features from XPath 2.0 and XPath 3.0.[1]

Note that the coverage of XPath 1.0, 2.0, and 3.0 given in Table 6.1 does not restrict function calls to the standard library [xfu, 2014]. The next column 'XPath 3.0 std' shows the coverage of XPath 3.0 when restricted to standard functions. We see here an essential limitation of analysing XPath queries in isolation, without support for non-standard functions, and in particular for user-defined functions: more than half of the queries extracted from XQuery documents are beyond the scope of our analyses.

The last column 'Core 2.0 extended' show the coverage of our extended version of the XPath fragment Core 2.0. The original fragment is presented in Chapter 7, and our extensions are presented in Chapter 8.

Functions. We show in Figure 6.5 the number of occurrences for each of the 400 most frequently occurring functions (among 1,600) and the associated accumulated percentage of the total number of function calls. Darker dots correspond to standard XPath functions, and the

[1]The output XQueryX representations of a handful of queries do not validate against XPath 3.0, due to out-of-bounds constant numerals; this is a very marginal effect.

```
<?xml version="1.0"?>
<benchmark>
<xpath column="45" filename="benchmark/example-histei.xqy" line="4">
  <query>data(($element/ attribute:: *[( id (.) is parent::node ())])[1]) </query>
  <ast size="30">
    <xqx:functionCallExpr xmlns:xqx="http://www.w3.org/2005/XQueryX">
      <xqx:functionName>data</xqx:functionName>
      <xqx:arguments>
        <xqx:pathExpr>
          <xqx:stepExpr>
            <xqx:filterExpr>
              <xqx:sequenceExpr>
                <xqx:pathExpr>
                  <xqx:stepExpr>
                    <xqx:filterExpr>
                      <xqx:varRef>
                        <xqx:name>element</xqx:name>
                      </xqx:varRef>
                    </xqx:filterExpr>
                  </xqx:stepExpr>
                  <xqx:stepExpr>
                    <xqx:xpathAxis>attribute</xqx:xpathAxis>
                    <xqx:Wildcard/>
                    <xqx:predicates>
                      <xqx:isOp>
                        <xqx:firstOperand>
                          <xqx:functionCallExpr>
                            <xqx:functionName>id</xqx:functionName>
                            <xqx:arguments>
                              <xqx:contextItemExpr/>
                            </xqx:arguments>
                          </xqx:functionCallExpr>
                        </xqx:firstOperand>
                        <xqx:secondOperand>
                          <xqx:pathExpr>
                            <xqx:stepExpr>
                              <xqx:xpathAxis>parent</xqx:xpathAxis>
                              <xqx:anyKindTest/>
                            </xqx:stepExpr>
                          </xqx:pathExpr>
                        </xqx:secondOperand>
                      </xqx:isOp>
                    </xqx:predicates>
                  </xqx:stepExpr>
                </xqx:pathExpr>
              </xqx:sequenceExpr>
            </xqx:filterExpr>
            <xqx:predicates>
              <xqx:integerConstantExpr>
                <xqx:value>1</xqx:value>
              </xqx:integerConstantExpr>
            </xqx:predicates>
          </xqx:stepExpr>
        </xqx:pathExpr>
      </xqx:arguments>
    </xqx:functionCallExpr>
  </ast>
</xpath>
</benchmark>
```

FIGURE 6.4. Example of output from our parser.

TABLE 6.1. The benchmark's list of sources.

Source	Queries	Coverage				
		XPath 1.0	XPath 2.0	XPath 3.0	XPath 3.0 std	Core 2.0 extended
DocBook http://docbook.sourceforge.net/	7,620	100.0%	100.0%	100.0%	95.6%	79.3%
TEIXSL https://github.com/TEIC/Stylesheets	6,303	96.4%	100.0%	100.0%	86.1%	70.8%
HTMLBook https://github.com/oreillymedia/HTMLBook	752	100.0%	100.0%	100.0%	92.0%	66.4%
Total (XSLT)	14,675	98.4%	100.0%	100.0%	91.3%	75.0%
XQuery parser https://github.com/jpcs/xqueryparser.xq	1,659	83.1%	85.2%	99.9%	15.7%	12.6%
eXist-db https://github.com/eXist-db	1,151	76.7%	88.0%	100.0%	64.8%	33.1%
HisTEI https://github.com/odaata/HisTEI	483	74.7%	97.5%	100.0%	62.5%	30.2%
transform.xq https://github.com/jpcs/transform.xq	365	64.3%	70.4%	99.7%	45.7%	20.5%
ml-enrich https://github.com/freshie/ml-enrich	302	74.1%	96.3%	100.0%	55.2%	39.4%
xquerydoc https://github.com/xquery/xquerydoc	269	87.3%	98.8%	100.0%	52.7%	28.9%
openinfoman https://github.com/openhie/openinfoman	261	65.1%	96.1%	100.0%	47.8%	26.8%
Oxford Dict API https://github.com/AdamSteffanick/od-api-xquery	207	85.5%	97.5%	100.0%	57.4%	55.5%
MarkLogic Commons https://github.com/marklogic/commons	196	70.9%	93.8%	97.4%	45.4%	24.4%
datascience https://github.com/adamfowleruk/datascience	184	77.1%	91.8%	100.0%	40.7%	21.1%
Link Management BaseX https://github.com/dita-for-small-teams/dfst-linkmgmt-basex/	154	72.0%	96.7%	100.0%	73.3%	36.3%
Semantic Web https://github.com/HeardLibrary/semantic-web/	149	86.5%	93.9%	100.0%	81.2%	51.0%
eXist annotation store https://github.com/telic/exist-annotation-store/	133	80.4%	86.4%	100.0%	64.6%	48.8%
xqtest https://github.com/irinc/xqtest/	130	70.0%	99.2%	100.0%	46.9%	38.4%
data.xq https://github.com/jpcs/data.xq/	119	32.7%	33.6%	100.0%	34.4%	16.8%
graphxq https://github.com/apb2006/graphxq/	92	73.9%	78.2%	100.0%	76.0%	47.8%
ml-invoker https://github.com/fgeorges/ml-invoker/	92	89.1%	89.1%	100.0%	36.9%	32.6%
treedown https://github.com/biblicalhumanities/treedown/	92	94.5%	97.8%	100.0%	80.4%	59.7%
XQJSON https://github.com/joewiz/xqjson/	90	74.4%	100.0%	100.0%	67.7%	55.5%
fots BaseX https://github.com/LeoWoerteler/fots-basex/	73	65.7%	71.2%	100.0%	63.0%	28.7%
GPXQuery https://github.com/dret/GPXQuery/	57	73.6%	98.2%	100.0%	64.9%	29.8%
rbtree.qx https://github.com/jpcs/rbtree.xq/	57	22.8%	28.0%	100.0%	24.5%	0%
xquery-libs https://github.com/adamretter/xquery-libs/	53	79.2%	88.6%	100.0%	60.3%	49.0%
Guid-O-Matic https://github.com/baskaufs/guid-o-matic/	51	84.3%	96.0%	100.0%	56.8%	41.1%
functional.xq https://github.com/jpcs/functional.xq/	47	12.7%	14.8%	100.0%	21.2%	2.1%
Total (XQuery)	6,466	76.1%	87.4%	99.8%	46.7%	28.0%
Total	21,141	91.6%	96.1%	99.9%	77.7%	60.6%

darker line corresponds to the accumulated percentage achieved by these standard functions. Arithmetic operations do not figure here as they are classified syntactically as *operators* in XPath. They occur more than the tenth most frequent function.

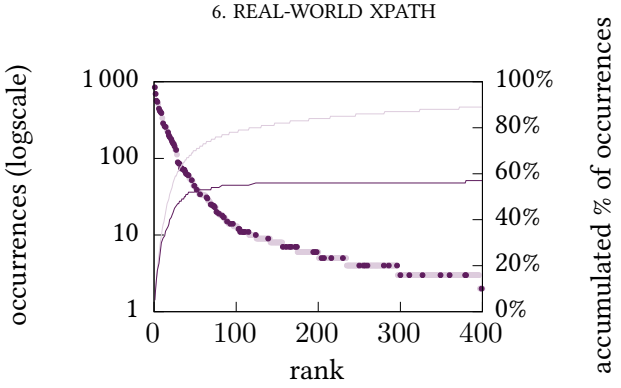

FIGURE 6.5. Occurrences of function calls.

The standard functions only represent 57.23% of the function calls in the benchmark. This is mostly due to queries from XQuery sources, which routinely use functions defined in the surrounding XQuery programs: when restricting to these sources, standard XPath functions represent 42.93% of the function calls. By contrast, when restricting to XSLT sources, we find only 210 functions and standard XPath functions represent 76.32% of the calls. Moreover, still within XSLT sources, the 16 functions with more than 100 occurrences each all belong to the XSLT or XPath standard, and account for 78.35% of the occurrences of function calls. In the XSLT sources, there are 4,650 queries (31.69%) performing at least one function call, roughly as many as the 4,556 queries (70.46%) found in the XQuery sources.

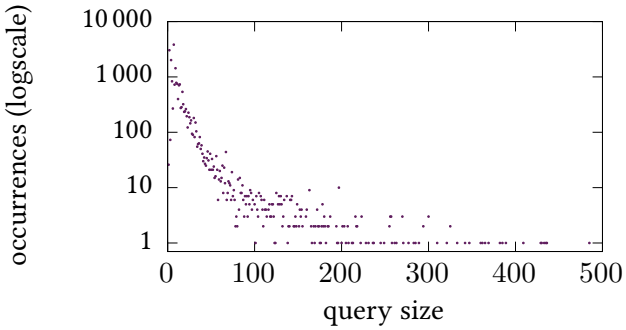

FIGURE 6.6. Distribution of query sizes.

Size. Figure 6.6 shows the distribution of query sizes, defined as the number of nodes in their syntax trees. As might be expected, a majority of the queries have size at most 13, but there are nevertheless 256 queries of size 100 or more.

6.3.4. Benchmark Occurrences. Table 6.2 shows the number of queries that use each specific axis, first for each type of source, and then globally; `attribute` and `child` are the most prominent axes.

Tables 6.3 and 6.4 show, for each syntactic construct, the number of queries that use it. Table 6.3 focuses on constructs from our restricted syntax, while Table 6.4 presents XPath constructs not supported by our abstract syntax. The difference between XSLT and XQuery sources is marked, with 'advanced' or unsupported constructs (`let`, `for`, `map`, etc.) used

TABLE 6.2. Number of queries using each axis.

Axis	XSLT	XQuery	Total
`ancestor(-or-self)`	753 (5.1%)	27 (0.4%)	780 (3.6%)
`attribute`	3,048 (20.7%)	578 (8.8%)	3,626 (17.1%)
`child`	5,479 (37.3%)	1,122 (17.2%)	6,601 (31.1%)
`descendant(-or-self)`	488 (3.3%)	288 (4.4%)	776 (3.6%)
`following`	18 (0.1%)	2 (0.0%)	20 (0.0%)
`following-sibling`	223 (1.5%)	18 (0.2%)	241 (1.1%)
`namespace`	10 (0.0%)	0 (0.0%)	10 (0.0%)
`parent`	522 (3.5%)	56 (0.8%)	578 (2.7%)
`preceding`	49 (0.3%)	3 (0.0%)	52 (0.2%)
`preceding-sibling`	237 (1.6%)	13 (0.2%)	250 (1.1%)
`self`	557 (3.7%)	16 (0.2%)	573 (2.7%)
All axes	8,351 (56.8%)	1,501 (23.0%)	9,852 (46.5%)

TABLE 6.3. Number of queries using each construct.

Syntactic construct	XSLT	XQuery	Total
$\alpha::*$	8,351 (56.8%)	1,501 (23.0%)	9,852 (46.5%)
$/\pi$	233 (1.5%)	39 (0.6%)	272 (1.2%)
π/π	3,355 (22.8%)	1,499 (23.0%)	4,854 (22.9%)
$\pi[\varphi]$	2,415 (16.4%)	1,070 (16.4%)	3,485 (16.4%)
π `union` π	1,155 (7.8%)	42 (0.6%)	1,197 (5.6%)
$f(\pi_1, \ldots, \pi_n)$	4,650 (31.6%)	4,555 (70.0%)	9,205 (43.4%)
`$x`	6,896 (46.9%)	5,580 (85.8%)	12,476 (58.9%)
`let $x :=` π `return` π	0 (0.0%)	405 (6.2%)	405 (1.9%)
`for $x in` π `return` π	5 (0.0%)	164 (2.5%)	169 (0.7%)
φ `or` φ, φ `and` φ	1,818 (12.3%)	234 (3.6%)	2,052 (9.6%)
π `is` π	1 (0.0%)	12 (0.1%)	13 (0.0%)
$\pi \triangle \pi$	545 (3.7%)	308 (4.7%)	853 (4.0%)
$\pi \triangle^+ d$	4,029 (27.4%)	762 (11.7%)	4,791 (22.6%)

almost only in XQuery, and navigation and data tests against constants used significantly more in XSLT.

6.3.5. Limitations. The benchmark is made of uncurated data, thus no distinction is made between tiny XPath queries and more interesting ones. For instance, only 9,852 of the queries in the benchmark use at least one axis step. Also, as seen in Table 6.1, the numbers of queries from the various sources are not balanced, which means that results on the whole benchmark might not be very telling, and that one should distinguish the XSLT sources from the XQuery ones. Finally, the benchmark was compiled specifically for investigating the coverage of syntactic fragments of XPath. It is currently not really suitable for other ends:

TABLE 6.4. Number of queries using unsupported syntactic constructs.

Unsupported construct	XSLT	XQuery	Total
simple map	0(0.0%)	44(0.6%)	44(0.2%)
dynamic function invocation	0(0.0%)	170(2.6%)	170(0.8%)
inline function	0(0.0%)	126(1.9%)	126(0.5%)
named function	0(0.0%)	31(0.4%)	31(0.1%)
range sequence	11(0.0%)	54(0.8%)	65(0.3%)
instance of	0(0.0%)	35(0.5%)	35(0.1%)
processing instruction	120(0.8%)	9(0.1%)	129(0.6%)
cast-related expressions	19(0.1%)	6(0.0%)	25(0.1%)

XPath satisfiability: in both XSLT and XQuery files, no schema information on the XML to be processed is available. Furthermore, it seems likely that most queries are satisfiable—quite possibly all of them.

XPath evaluation: similarly, the benchmark does not provide examples of input XML documents on which the XSLT or XQuery should be evaluated.

The first limitation could be lifted by inspecting each source and manually adding the relevant schema when it can be identified.

6.3.6. Related Work. Regarding XPath and XQuery, the previous works [e.g. Afanasiev and Marx, 2008; Franceschet, 2005; Schmidt et al., 2002] on benchmarks focus on evaluating the performance of processors. For instance, Franceschet [2005] comprises two collections of queries: functional queries (XPathMark-FT) check the functional correctness of XPath processors, while performance queries (XPathMark-PT) allow to evaluate the performance of query evaluation in XPath processors. These benchmarks are synthetic and of limited size, and accompanied with XML documents against which they should be evaluated. Compared to these works, we carry a large scale analysis of real-world queries, and analyse them with respect to the satisfiability problem rather than the evaluation problem. We did not include these synthetic benchmarks in our analysis, as we focus on real-world queries.

Several large scale studies of real-world SPARQL queries harvest from semantic web search logs [Bonifati et al., 2017; Picalausa and Vansummeren, 2011]. Thanks to the availability of SPARQL logs, the latest one [Bonifati et al., 2017] includes over 50 million unique queries and carries out detailed analyses of query features that are relevant to their evaluation, including whether the queries belong to specific SPARQL fragments. While we did not focus on XPath fragments for which evaluation would be more efficient, our benchmark could certainly be exploited in this direction.

CHAPTER 7

Decidable XPath Fragments

Contents

7.1. Introduction

We now present the fragments with decidable satisfiability and containment we have considered in our experiments. As there is such an abundant literature on the topic [e.g. Benedikt and Koch, 2009; Benedikt et al., 2008; ten Cate and Lutz, 2009; Figueira, 2012a,b, 2013; Figueira and Segoufin, 2017; Gottlob and Koch, 2002; Jurdziński and Lazić, 2011; Neven and Schwentick, 2006; Schwentick, 2004], this is clearly an incomplete sample, but we think it is representative of the main lines of investigation.

The fragments we consider in our experiments and their inclusions are shown in Figure 7.1, along with the complexity of satisfiability in each fragment. Regarding complexity, we use the DAG-size of the input expression, where isomorphic sub-expressions are shared. Note that Figure 7.1 reports the complexity for the original logics, thus for EMSO^2 and non-mixing MSO constraints, the complexity of the XPath fragments we translate into the logics might be lower.

7.2. Decidable XPath Fragments

7.2.1. Positive XPath. Some of the earliest-studied fragments of XPath are based on tree patterns [Hidders, 2004]. Geerts and Fan [2005, Thm. 4] show that the following PositiveXPath

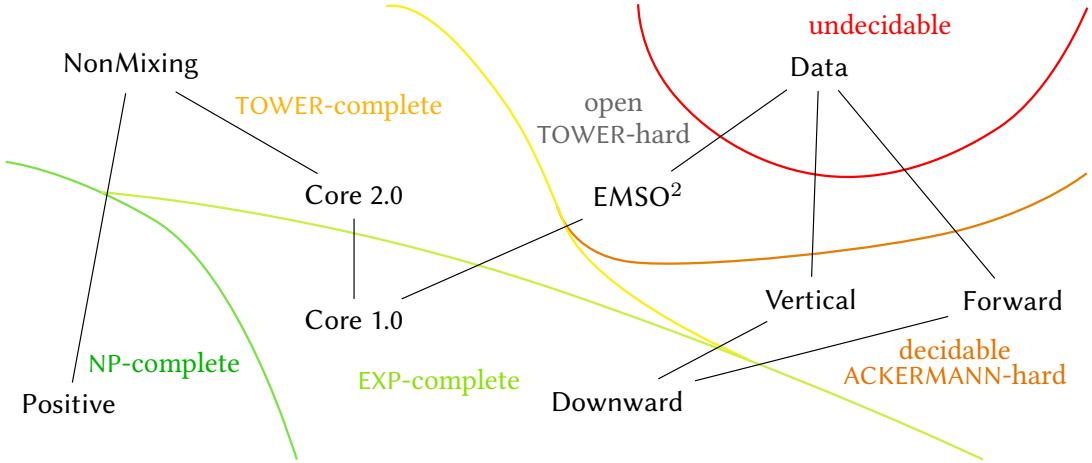

FIGURE 7.1. Inclusions and complexities of the fragments of Chapter 7.

fragment has an NP-complete satisfiability problem, even in presence of a DTD,

$$\pi ::= \alpha{::}* \mid \pi/\pi \mid \pi[\varphi] \mid \pi \texttt{ union } \pi \mid \pi \texttt{ intersect } \pi$$
$$\varphi ::= \pi \mid a \mid \texttt{false}() \mid \texttt{true}() \mid \varphi \texttt{ or } \varphi \mid \varphi \texttt{ and } \varphi \mid \pi \triangle \pi$$

where a ranges over Σ and $\triangle$ over $\{\texttt{eq}, \texttt{ne}\}$. This fragment has a semantics-preserving translation into the existential positive fragment of first-order logic over the relational signature $(\downarrow, \downarrow^*, \rightarrow, \rightarrow^*, (P_a)_{a\in\Sigma}, \sim, \not\sim)$.

7.2.2. Core XPath 1.0. A landmark fragment is the language of Gottlob and Koch [2002], known in the literature as 'CoreXPath'. This language is akin to propositional dynamic logic on trees [Afanasiev et al., 2005], and is defined by the abstract syntax

$$\pi ::= \alpha{::}* \mid \pi/\pi \mid \pi[\varphi] \mid \pi \texttt{ union } \pi$$
$$\varphi ::= \pi \mid a \mid \texttt{false}() \mid \texttt{not}(\varphi) \mid \varphi \texttt{ or } \varphi$$

where a ranges over Σ. We recall here the usual *standard translation* of CoreXPath expressions into first-order formulæ [see, e.g., Afanasiev et al., 2005; Marx, 2005]:

$$\begin{aligned}
\mathrm{ST}_{x,y}(\alpha{::}*) &= x \; [\![\alpha]\!]_{\mathsf{A}} \; y \\
\mathrm{ST}_{x,y}(\pi/\pi') &= \exists z \,.\, \mathrm{ST}_{x,z}(\pi) \wedge \mathrm{ST}_{z,y}(\pi') && (z \text{ fresh}) \\
\mathrm{ST}_{x,y}(\pi[\varphi]) &= \mathrm{ST}_{x,y}(\pi) \wedge \mathrm{ST}_{y}(\varphi) \\
\mathrm{ST}_{x,y}(\pi \texttt{ union } \pi') &= \mathrm{ST}_{x,y}(\pi) \vee \mathrm{ST}_{x,y}(\pi') \\
\mathrm{ST}_{x}(\pi) &= \exists y \,.\, \mathrm{ST}_{x,y}(\pi) && (y \text{ fresh}) \\
\mathrm{ST}_{x}(a) &= a(x) \\
\mathrm{ST}_{x}(\texttt{false}()) &= \bot \\
\mathrm{ST}_{x}(\texttt{not}(\varphi)) &= \neg \mathrm{ST}_{x}(\varphi) \\
\mathrm{ST}_{x}(\varphi \texttt{ or } \varphi) &= \mathrm{ST}_{x}(\varphi) \vee \mathrm{ST}_{x}(\varphi')
\end{aligned}$$

Furthermore, CoreXPath is known to be expressively equivalent to the two variable fragment of the first order logic on trees [Marx and De Rijke, 2005].

CoreXPath has an EXP-complete satisfiability problem, also in presence of a DTD. To the best of our knowledge, this is the only fragment in this section for which an implementation of a satisfiability procedure exists [Genevès et al., 2015].

7.2.3. Core XPath 2.0. The main feature of XPath 2.0 was the introduction of `for` loops. Having `for` loops further enriches XPath with variable quantification, and provides a significant jump in expressiveness (see §6.2.5.2). enTEN Cate and Lutz [2009] study the extension of CoreXPath with

$$\pi ::= \cdots \mid \$\mathtt{x} \mid \mathtt{for}\ \$\mathtt{x}\ \mathtt{in}\ \pi\ \mathtt{return}\ \pi$$
$$\varphi ::= \cdots \mid \$\mathtt{x}\ \mathtt{is}\ \$\mathtt{y} \mid .\ \mathtt{is}\ \$\mathtt{x}$$

where $x and $y range over $\mathcal{X}$; we call the resulting fragment CoreXPath 2.0. The syntax of node identity tests in ten Cate and Lutz [2009] is slightly more restrictive than ours, but this can be fixed by defining $\pi\ \mathtt{is}\ \pi'$ as a shorthand for

$$\textit{singleton}(\pi)\ \text{and}\ \textit{singleton}(\pi')\ \text{and}\ \mathtt{for}\ \$\mathtt{x}\ \mathtt{in}\ \pi\ \mathtt{return}\ \mathtt{for}\ \$\mathtt{y}\ \mathtt{in}\ \pi'\ \mathtt{return}\ .[\$\mathtt{x}\ \mathtt{is}\ \$\mathtt{y}] \quad (7.1)$$

A deeper difference lies in the semantics of variables: ten Cate and Lutz [2009] assume that valuations map to single nodes. This does not make any difference regarding bound variables, since in CoreXPath 2.0 they must be bound by a `for` expression, but it does make one for free variables. This is not an issue, since the decision procedure for CoreXPath 2.0 can handle those.

Indeed, satisfiability in CoreXPath 2.0 is decidable in time bounded by a tower of exponentials, whose height depends on the size of the expression. This is seen by reducing the problem to satisfiability in $\mathrm{MSO}(\downarrow, \rightarrow, (P_a)_{a\in\Sigma})$, using the standard translation of CoreXPath expressions into MSO formulæ recalled in Section 7.2.2; due to the presence of variables, this translation must be refined with a mapping τ from XPath variables to first-order variables, allowing to write

$$\begin{aligned}
\mathrm{ST}^{\tau}_{x,y}(\$\mathtt{z}) &= (y = \tau(\$\mathtt{z})) \\
\mathrm{ST}^{\tau}_{x,y}(\mathtt{for}\ \$\mathtt{z}\ \mathtt{in}\ \pi\ \mathtt{return}\ \pi') &= \exists z\,.\,\mathrm{ST}^{\tau}_{x,z}(\pi) \wedge \mathrm{ST}^{\tau[\$\mathtt{z}\mapsto z]}_{x,y}(\pi') \qquad (z \text{ fresh}) \\
\mathrm{ST}^{\tau}_{x}(\$\mathtt{y}\ \mathtt{is}\ \$\mathtt{z}) &= (\tau(\$\mathtt{y}) = \tau(\$\mathtt{z})) \\
\mathrm{ST}^{\tau}_{x}(.\ \mathtt{is}\ \$\mathtt{z}) &= (x = \tau(\$\mathtt{z}))
\end{aligned}$$

The resulting TOWER complexity upper bound is tight [ten Cate and Lutz, 2009, Thm. 31]. Thus our free XPath variables translate directly into free second-order variables in $\mathrm{MSO}(\downarrow, \rightarrow, (P_a)_{a\in\Sigma})$.

7.2.4. Data XPath. A well-studied XPath fragment with the ability to test data equality and disequality is DataXPath [Geerts and Fan, 2005]. It is obtained by adding *joins* to the syntax of CoreXPath as follows, where $\triangle$ ranges over $\{\mathtt{eq}, \mathtt{ne}\}$:

$$\varphi ::= \cdots \mid \pi \triangle \pi$$

Although DataXPath satisfiability is undecidable [Geerts and Fan, 2005], restricting the navigational power restores decidability [Figueira, 2018]. The first decidable fragment we consider is VerticalXPath, shown decidable by Figueira and Segoufin [2017, Thm. 2.1], which restricts the syntax of DataXPath to only allow vertical navigation:

$$\alpha ::= \texttt{self} \mid \texttt{child} \mid \texttt{descendant} \mid \texttt{parent} \mid \texttt{ancestor}$$

Another decidable fragment is ForwardXPath, shown decidable by Figueira [2012b, Thm. 6.4], where navigation is restricted to forward axes only:

$$\alpha ::= \texttt{self} \mid \texttt{child} \mid \texttt{descendant} \mid \texttt{following-sibling}$$

We also consider DownwardXPath [Figueira, 2012a, Thm. 6.4], the intersection of VerticalXPath and ForwardXPath, where only downward navigation is allowed:

$$\alpha ::= \texttt{self} \mid \texttt{child} \mid \texttt{descendant}$$

As seen in Figure 7.1, the complexity of the satisfiability problem in these three fragments varies considerably: DownwardXPath is EXP-complete, but VerticalXPath and ForwardXPath are ACKERMANN-hard [Figueira and Segoufin, 2009]. It is also notable that satisfiability of DownwardXPath also becomes ACKERMANN-hard in presence of DTDs [Figueira and Segoufin, 2009].

7.2.5. Existential MSO^2. Bojańczyk, Muscholl, Schwentick, and Segoufin [2009] investigate the satisfiability of formulæ of the form $\exists X_1 \cdots \exists X_n \,.\, \psi$, where $X_1, \ldots, X_n$ are monadic second-order variables and ψ is a first-order formula in the two-variable fragment over either

- the signature $(\downarrow, \rightarrow, (P_a)_{a\in\Sigma}, \sim)$, which they denote by EMSO$^2(\sim, +1)$, or
- the signature $(\downarrow, \downarrow^+\!\!\rightarrow, \rightarrow^+, (P_a)_{a\in\Sigma}, \sim)$, which they denote by EMSO$^2(\sim, <, +1)$.

In the first instance, they prove the decidability of satisfiability in 3-NEXP [Bojańczyk et al., 2009, Thm. 3.1], while the best known lower bound is NEXP-hardness, which holds already for first-order logic with two variables FO$^2(\downarrow, (P_a)_{a\in\Sigma})$ [Benaim et al., 2016, Thm. 5.1]. In the second instance, decidability is open, and equivalent to the reachability problem in an extension of branching vector addition systems [Jacquemard et al., 2016], with a TOWER lower bound [Lazić and Schmitz, 2015].

These results can be exploited for a fragment EMSO^{2}XPath of DataXPath: Bojańczyk et al. [2009, Thm. 6.1] allow the following restricted joins in CoreXPath

$$\pi ::= \cdots \mid \pi \triangle /\pi \mid \,.\, \triangle \alpha{::}*[\varphi]$$

where $\triangle$ ranges over $\{\texttt{eq}, \texttt{ne}\}$. When the above π, π', and α are restricted to using the axes `self`, `child`, and `parent`, this can be translated into EMSO$^2(\sim, +1)$, and the general form into EMSO$^2(\sim, <, +1)$.[1] In spite of the unknown decidability status of EMSO$^2(\sim, <, +1)$, we have run our benchmarks against the full logic.

[1] Bojańczyk et al. [2009] actually also allow joins of the form $@a \triangle \alpha{::}*/@b$, but this is subject to a semantic condition.

7.2.6. Non-Mixing MSO Constraints. In [Czerwinski et al., 2017], Czerwinski, David, Murlak, and Parys define *MSO constraints* as formulæ of the form $\psi(\bar{x}) \implies \eta_{\sim}(\bar{x}) \wedge \eta_{\not\sim}(\bar{x})$, where ψ is an MSO formula over the signature $(\downarrow, \rightarrow, (P_a)_{a\in\Sigma})$ with the first-order variables $\bar{x}$ as its free variables, and $\eta_{\sim}$ and $\eta_{\not\sim}$ are positive Boolean combinations of atoms, over the respective signatures $(\sim, (P_d)_{d\in\mathbb{D}})$ and $(\not\sim, (\neg P_d)_{d\in\mathbb{D}})$. Hence data tests and data joins are permitted, as long as they are not *mixed*. Satisfiability is called 'consistency' in this context, and is decidable [Czerwinski et al., 2017, Thm. 4]; better complexities are achievable when restricting ψ to conjunctive queries.

To quote Czerwinski et al. [2017], their 'results imply decidability... of the containment problem in the presence of a schema for unions of XPath queries without negation, where each query uses either equality or inequality, but never both.' Here is indeed a NonMixingXPath fragment of XPath, which can be translated to MSO constraints:

$$\varphi_c ::= \varphi_{\mathrm{eq}} \mid \varphi_{\mathrm{ne}} \mid \varphi_c \texttt{ or } \varphi_c$$

where $\triangle$-*expressions*, for $\triangle$ in $\{\mathtt{eq}, \mathtt{ne}\}$, are defined by

$$\begin{aligned}\pi_\triangle &::= \alpha{::}* \mid \pi_\triangle/\pi_\triangle \mid \pi_\triangle[\varphi_\triangle] \mid \pi_\triangle \texttt{ union } \pi_\triangle\\ \varphi_\triangle &::= \texttt{true()} \mid \texttt{false()} \mid \pi_\triangle \mid \varphi \mid \varphi_\triangle \texttt{ or } \varphi_\triangle \mid \varphi_\triangle \texttt{ and } \varphi_\triangle\\ &\quad \mid \pi_\triangle \triangle \pi_\triangle \mid \pi_\triangle \triangle d\end{aligned}$$

where φ is any CoreXPath 2.0 node expression. As this fragment embeds CoreXPath 2.0, its satisfiability problem is TOWER-hard, which matches the upper bound for MSO constraints [Czerwinski et al., 2017].

We now show how to translate a formula from this XPath fragment to a formula of the fragment of Czerwinski et al. [2017], that is a formula of the form $\bigvee_i(\exists \bar{x}_i \,.\, \alpha_i(\bar{x}_i) \wedge \eta^i_\triangle(\bar{x}_i))$, where the α_i are MSO formulæ and $\eta^i_\triangle$ positive Boolean combinations of $\triangle$ tests. The general idea of this translation is to first describe the tree structure of a tree satisfying the query (by using enough variables), which will provide the formulæ α_i, and then state all the data constraints that should hold between these variables, which will yield the formulæ $\eta^i_\triangle$.

The translation extends the standard translation for CoreXPath 2.0 with a top-level translation for $\triangle$-expressions:

$$\mathrm{ST}_x(\pi_\triangle \triangle \pi'_\triangle) = \exists y \exists z \,.\, \mathrm{ST}_{x,y}(\pi_\triangle) \wedge \mathrm{ST}_{x,z}(\pi'_\triangle) \wedge y \triangle z \qquad (y, z \text{ fresh})$$

Since this last case must occur positively, we will always be able to extract the atoms of the form $y \triangle z$ and regroup them in a formula $\eta_\triangle$.

7.3. Baseline Benchmark Results

We have implemented the fragments of this section as Relax NG schemas (see the files `relaxng/xpath-FRAGMENT-orig.rnc` in the distribution). In each case we included obvious extensions, such as the syntactic sugars discussed in Section 6.2.5 (in particular, the `last()` function is included in CoreXPath 2.0 and NonMixingXPath).

The results of these fragments on the benchmark are presented in Figures 7.2 and 7.3. The fragments allowing free variables, namely CoreXPath 2.0, EMSO^{2}XPath, and NonMixingXPath, have the best baseline coverage. We see here the practical interest of a fragment like NonMixingXPath with restricted negation but some support for variables, data tests $\pi \triangle d$, and data joins $\pi \triangle \pi$. The other fragments have an essentially negligible coverage in the

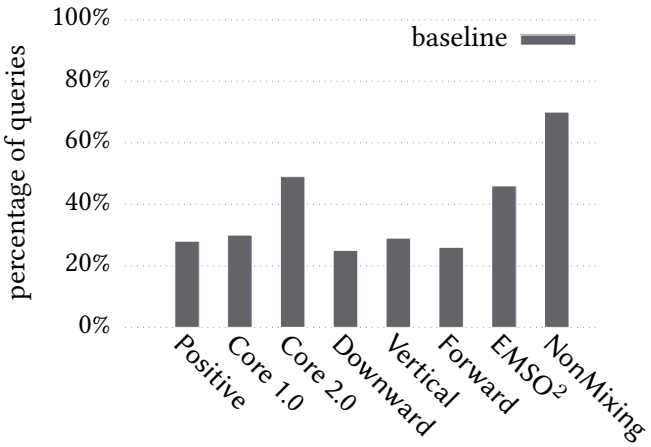

FIGURE 7.2. Coverage of the XSLT sources for the baseline fragments.

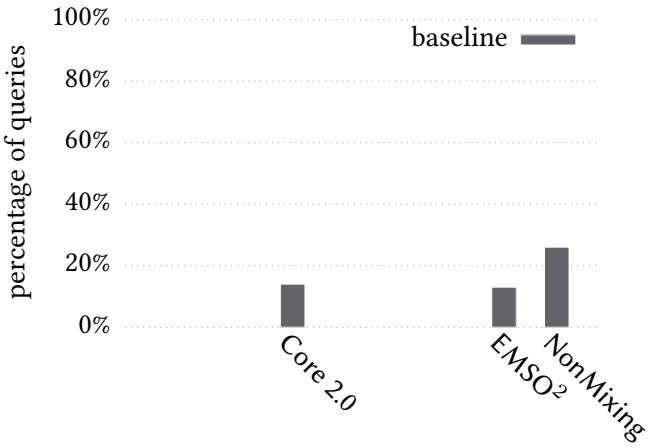

FIGURE 7.3. Coverage of the XQuery sources for the baseline fragments.

XQuery benchmarks (Figure 7.3). The support for unrestricted joins $\pi \triangle \pi'$ in the fragments of DataXPath from Section 7.2.4 has a very limited effect, and indeed we only found 65 relevant instances in the entire benchmark, i.e. where neither π nor π' is a variable or a function call.

Of course, the fragments defined in the literature were not meant to be run against concrete XPath queries; and the definitions we picked were somewhat arbitrary—e.g. some papers would allow root navigation $/\pi$ in CoreXPath 1.0. Hence, we do not consider these initial results as very significant. We will see in the next chapter that several easy extensions can be made to our fragments to reflect their expressivity more faithfully. The coverage of the extended versions of the fragments is presented and discussed in Section 8.5.

CHAPTER 8

Extensions

Contents

8.1. Introduction

In this chapter, we introduce several extensions of the fragments from Chapter 7, while preserving the decidability and complexity of the corresponding satisfiability problems. As seen in Table 8.1, we consider first 'basic' extensions with considerable impact on the benchmark coverage in Sections 8.2.1 to 8.2.3, and then 'advanced' ones with smaller impact in Sections 8.3.1 to 8.3.3.

We use two ways to prove that an extension can be handled: for any expression of the extended fragment, we either provide an *equivalent* or an *equisatisfiable* formula in the original fragment.

Definition 8.1. We say that an extension can be *expressed* in a fragment if, for any node expression φ in the extended syntax, we can compute an *equivalent* node expression φ' in the original fragment, i.e. such that for all data trees and valuations ν, $[\![\varphi]\!]^\nu_{\mathsf{N}} = [\![\varphi']\!]^\nu_{\mathsf{N}}$—and similarly for path expressions. Moreover, we say that the extension can be *polynomially encoded* if φ' can be computed in polynomial time.

Table 8.1. Occurrence counts of the syntactic extensions of Chapter 8 in the entire benchmark.

	Basic		Advanced		
$/\pi$	`$x`	$\pi \triangle^+ d$	$\pi \triangle \pi$	`last()`	`id()`
272	12,476	4,791	853	1,203	31

Definition 8.2. We say that an extension can be *encoded* in a fragment if, for any node expression φ in the extended syntax, we can compute an *equisatisfiable* node expression φ' in the original fragment, i.e. such that there exists t with $t \models \varphi$ if and only if there exists t' with $t' \models \varphi'$. Moreover, we say that the extension can be *polynomially encoded* if φ' can be computed in polynomial time.

Clearly, an extension that can be (polynomially) expressed can also be (polynomially) encoded, but the converse might not hold.

Last of all, the proof techniques employed to show the decidability of satisfiability in a fragment might allow to handle the extension at hand.

8.2. Basic Extensions

8.2.1. $/\pi$: Root Navigation. In XPath, navigation to the root is possible through the $/\pi$ construct as well as using the nullary $\texttt{root}()$ function [xfu, 2014], with semantics $[\![\texttt{root}]\!]_{\mathcal{F}} = \{\varepsilon\}$.

We naturally allow these features in CoreXPath 1.0 and 2.0 as well as in the NonMixingXPath and EMSO^{2}XPath fragments, where navigation to the root is captured by

$$\texttt{ancestor-or-self::}*[\texttt{not}(\texttt{parent::}*)] \tag{8.1}$$

The same goes for VerticalXPath but not for the two other DataXPath fragments, where one cannot navigate upwards. It is clear that root navigation is not expressible in these fragments. In fact, it cannot even be encoded in ForwardXPath, since it becomes undecidable when extended with navigation to the root, as can be seen by adapting the proofs from Figueira and Segoufin [2009].

Proposition 8.3. *Satisfiability in ForwardXPath extended with root navigation is undecidable.*

Proof. This is similar to the proof of Cor. 4 in Figueira and Segoufin [2009].

Their Thm. 2 shows the ACKERMANN-hardness of the satisfiability of simple 1-register freeze LTL over data words, with strict future temporal modalities (denoted by $\mathsf{sLTL}_1^{\downarrow}(\mathsf{F}_s)$). As explained in their Prop. 1, any formula from this fragment of freeze LTL can be translated into an equivalent XPath formula. In ForwardXPath, we can force the data trees to consist of a single branch, i.e. to be data words.

As seen in the proof of their Thm. 3, the only reason ForwardXPath is 'only' ACKERMANN-hard instead of undecidable is that, in their construction in Thm. 2 of formulæ simulating runs of Minsky counter machines, one cannot check in ForwardXPath that every decrement was preceded by a matching increment higher in that tree. But we can check that no matching increment occurs down that point with

$$\texttt{not}\big(\texttt{//.}\big[\textsc{dec}(i)\ \texttt{and}\ (\texttt{. eq .//.}[@])\big]\big) \tag{8.2}$$

and thus the following ensures that any decrement has a matching increment closer to the root

$$\texttt{not}\big(\texttt{//.}\big[\textsc{dec}(i)\ \texttt{and not}(\texttt{. eq //.}[@])\big]\big) \tag{8.3}$$

where $\textsc{dec}(i)$ and $@$ are labels in Σ introduced in their construction. □

We leave open the question whether there is a (polynomial) encoding of root navigation in DownwardXPath.

Finally, regarding PositiveXPath, Hidders's original fragment [Hidders, 2004] allowed root navigation, and his proof of a small model property (showing the satisfiability problem to be in NP) applies mutatis mutandis to the fragment with data joins defined in Section 7.2.1.

8.2.2. $x: Free Variables. The XPath specification mentions that all variables are essentially second-order. More precisely, a variable is interpreted as an ordered collection of items which may be nodes or data values. In practice, queries extracted from XQuery or XSLT applications contain variables that are bound by the host language. They may be bound to nodes, node collections, or data values. It is out of the scope of the present work to recover such information to consider a more specific satisfiability problem. Rather, we interpret all variables as unordered collections of nodes, as can be seen in our semantics.

In most of our fragments, free variables are admissible. Indeed, any formula φ over Σ and $\mathbb{D}$ with a (necessarily finite) set of free variables $X \subseteq \mathcal{X}$ can be translated into an equisatisfiable formula φ^X over $\Sigma \times 2^X$ and $\mathbb{D}$ with no free variables. Let us write a_S for $(a, S) \in \Sigma \times 2^X$. The key translation steps are:

$$(\$\mathrm{x})^X = //.\big[\bigvee_{a\in\Sigma,\, \$\mathrm{x}\in S\subseteq 2^X} a_S\big] \qquad\qquad (a)^X = \bigvee_{a\in\Sigma,\, S\subseteq 2^X} a_S \tag{8.4}$$

Assuming without loss of generality that the variables bound by constructs such as `for` or `let` do not belong to X, they are not affected by this translation. Given a data tree $t = (\ell, \delta)$ and a valuation $\nu\colon X \mapsto N$, we define $t^\nu = (\ell^\nu, \delta)$ by $\ell^\nu(p) = \ell(p)_{\{\$\mathrm{x}\in X \mid p\in\nu(\$\mathrm{x})\}}$. We then have $t, p, \nu \models \varphi$ if and only if $t^\nu, p \models \varphi^X$, and similarly for path expressions. Since any data tree t' over $\Sigma \times 2^X$ is of the form t^ν for some ν, we conclude that φ and φ^X are equisatisfiable.

Proposition 8.4. *Free variables can be encoded in PositiveXPath, CoreXPath 1.0, CoreXPath 2.0, VerticalXPath, NonMixingXPath, and EMSO²XPath.*

Note that although the encoding is exponential, the extension does not actually impact the complexity of satisfiability: in PositiveXPath and CoreXPath 1.0, a polynomial encoding can be obtained, leveraging a variant of the semantics where multiple propositions may hold at a node. In VerticalXPath, satisfiability is ACKERMANN-hard, so an exponential blow-up will not have an effect on the worst-case complexity. The decision procedures for the fragments CoreXPath 2.0, NonMixingXPath, and EMSO²XPath are based on second-order logic, hence they actually allow XPath variables in their baseline version.

We finally observe that this translation is not available in DownwardXPath and ForwardXPath, because they cannot express the root path of (8.4). In fact, free variables cannot be encoded in ForwardXPath; this is similar to Proposition 8.3.

Proposition 8.5. *Satisfiability in ForwardXPath extended with one free variable is undecidable.*

Proof. We reduce the satisfiability of ForwardXPath + `root()` over data trees with a single branch to the satisfiability of ForwardXPath with a single variable, which is therefore undecidable by Proposition 8.3.

Let $\mathrm{r} be the free variable, and let r be a fresh label not found in Σ. We first ensure that all the nodes in the valuation of $\mathrm{r} have r as label, with the constraint not($\mathrm{r}[not($r$)]).

Since we work with data trees consisting of a single branch, there is a lowest node in $\nu(\$\mathtt{r})$: this is the node selected uniquely by the path expression $\$\mathtt{r}[\mathtt{not}(.//r)]$. We use it as our root and perform the entire construction of [Figueira and Segoufin, 2009, Thm. 3] and Proposition 8.3 below that node; (8.3) becomes

$$\mathtt{not}\big(\$\mathtt{r}[\mathtt{not}(.//r)]//.\big[\textsc{dec}(i)\text{ and }\mathtt{not}(.\ \mathtt{eq}\ //.[@])\big]\big) \qquad \square$$

8.2.3. $\pi\triangle^+ d$: Data Tests against Constants. We now consider the extension with direct comparisons against constants from $\mathbb{D}$, assuming that $<$ is a dense total order:[1]

$$\varphi ::= \cdots \mid \pi\ \triangle^+\ d$$

where $\triangle^+ \in \{\mathtt{eq}, \mathtt{ne}, \mathtt{le}, \mathtt{lt}, \mathtt{ge}, \mathtt{gt}\}$ and $d \in \mathbb{D}$.

Going slightly further, we allow comparisons against constant data *expressions* in our fragments, where constant expressions are built from constant values and the deterministic context-insensitive functions of the XPath specification [xfu, 2014], i.e., function calls that can be precomputed before testing the query for validity. For instance, @n eq $3 + 1$ or $@a =$ concat('foo','bar') are allowed, but @n eq $@m + 1$ and $@a =$ concat('foo',$@b$) are not.

Proposition 8.6. *Data tests can be polynomially encoded in CoreXPath 1.0 and 2.0, Vertical, Downward, Forward and EMSO^2 XPath.*

We now sketch how any formula over Σ and $\mathbb{D}$ featuring comparisons against constants in a finite subset $D \subseteq \mathbb{D}$ can be transformed into an equisatisfiable formula over an extended labelling set $\Sigma \times C_D$. This is similar to the treatment of free variables in Section 8.2.2, but requires to include data consistency constraints in the encoded formula when the fragment at hand supports data joins. Crucially, these consistency constraints are mixing, and thus not available in NonMixingXPath. Regarding PositiveXPath, the small model property [Hidders, 2004, Lem. 1] still holds in the presence of data tests, hence satisfiability remains in NP for this extension.

Consider expressions of any of the considered fragments, extended with data tests against constants taken in a finite subset $D \subseteq \mathbb{D}$. We assume that $(\mathbb{D}, <)$ is dense and unbounded[2]. For convenience, we assume without loss of generality that $D = \{d_1, \ldots, d_n\}$ with $d_i < d_j$ when $i < j$.

We first define a translation $(\cdot)^D$ which maps node and path expressions over Σ with data tests in D to expressions without data tests but over

$$\Sigma_D = \Sigma \times C_D \text{ with } C_D = \{-\infty, +\infty\} \cup D \cup \{(d_i, d_{i+1}) \mid 1 \leq i < n\}.$$

As before, compound labels $(a, c) \in \Sigma_D$ are written a_c. Intuitively, the extra information will classify the value x held by a node: either $x < d_1$, or $d_n < x$, or $x = d_i$ or $d_i < x < d_{i+1}$ for

[1] These assumptions are not met in the actual XPath model where comparisons are undefined between numeric and arbitrary string values, but this problem can be avoided e.g. when the type of attributes is known from a schema.

[2] Our argument can easily be adapted to work with the unboundedness assumption: $-\infty$ should simply be dropped from C_D when d_1 is minimal, and similarly for $+\infty$ when d_n is maximal. For target fragments without data joins, the density assumption can be dropped if we also remove (d_i, d_{i+1}) from C_D when $[(d_i, d_{i+1})] = \emptyset$. To get rid of density in other fragments, we would need to know when there exists only finitely many values in some $[c]$ for $c \in C_D$ *and* could express in XPath that nodes with this comparison tag should not carry more than this many distinct values. Unfortunately, the latter does not seem to be feasible.

some i. It will be convenient to define, for any $d_i \in D$,

$$C_D^{<d_i} = \{-\infty\} \cup \{d_j \mid j < i\} \cup \{(d_j, d_{j+1}) \in C_D \mid j < i\}.$$

We give below the key translation steps:

$$\begin{aligned}
(a)^D &= \operatorname*{or}_{d \in C_D} a_d \\
(\pi \text{ eq } d)^D &= (\pi)^D \big[\operatorname*{or}_{a \in \Sigma} a_d\big] \\
(\pi \text{ ne } d)^D &= (\pi)^D \big[\operatorname*{or}_{a \in \Sigma} \operatorname*{or}_{c \in C_D, c \neq d} a_c\big] \\
(\pi \text{ lt } d)^D &= (\pi)^D \big[\operatorname*{or}_{a \in \Sigma} \operatorname*{or}_{c \in C_D^{<d}} a_c\big]
\end{aligned}$$

The cases of gt, le and ge are similar. The translation is homomorphic w.r.t. all other constructs including data joins ($\pi \triangle \pi'$ with $\triangle \in \{\text{eq}, \text{ne}\}$).

8.2.3.1. *Equisatisfiability for Data-Consistent Trees.* Define for all $c \in C_D$ its interpretation $[c] \subseteq \mathbb{D}$ in the natural way:

$$[-\infty] = \{d \in \mathbb{D} \mid d < d_1\}, \quad [d] = \{d\}, \quad [(d_i, d_{i+1})] = \{d \in \mathbb{D} \mid d_i < d < d_{i+1}\}, \quad \text{etc.}$$

Given a data tree $t = (\ell, \delta)$, we define $t^D = (\ell^D, \delta)$ with $\ell^D(p) = \ell(p)_c$ when $\delta(p) = d \in D$ and c is the unique element of C_D such that $d \in [c]$. We say that trees of the form t^D are data-consistent, because their data values respect their comparison tags.

Proposition 8.7. *For all φ with data tests in D, for all t and p, $t, p \models \varphi$ iff $t^D, p \models \varphi^D$ (and similarly for path expressions).*

Proof. Immediate by induction over expressions. □

8.2.3.2. *Equisatisfiability.* In CoreXPath 1.0 and CoreXPath 2.0, the encoded formula φ^D is insensitive to data values, hence any model t' of φ^D can be modified into a model of the form t^D by changing the data values according to the compound labels. This shows that φ and φ^D are equisatisfiable in these fragments.

For the fragments with data joins under consideration, i.e. VerticalXPath, DownwardXPath, and ForwardXPath, we show that φ is equisatisfiable with the following formula:

$$(\varphi)^D \text{ and } \text{not}\big(\operatorname*{or}_{d \in D,\, a,b \in \Sigma} //a_d \text{ ne } //b_d\big) \text{ and } \text{not}\big(\operatorname*{or}_{a,b \in \Sigma} \operatorname*{or}_{c \neq c' \in C_D} //a_c \text{ eq } //b_{c'}\big) \tag{8.5}$$

The above translation is obviously in VerticalXPath when $(\varphi)^D$ is in that fragment. For DownwardXPath and ForwardXPath, because they cannot visit any node above the initial evaluation point and do not allow free variables, it is safe to allow expressions of the form $\text{not}(//.[\text{not}(\varphi)])$ and φ' (like (8.5)), meaning that φ holds everywhere in the tree and φ' at the point of evaluation. Indeed, this is equivalent in these fragments to $\text{not}(.//.[\text{not}(\varphi)])$ and φ'.

Assuming that this formula admits a model, we show that it has a model of the form t^D, i.e. a model in which data values are consistent with compound labels. We use the fact that modifying the data values of a model in an injective way yields another model, because $(\varphi)^D$ only performs (dis)equality tests on data (through $\pi \triangle \pi'$ constructs):

- Thanks to the extra constraints in our formula we know that, for all $d \in D$, all nodes with a tag in $\Sigma \times \{d\}$ have the same value, and that this value is not present in other

nodes of the tree. Thus we can assume without loss of generality that we have a model t' such that, for any node n of t' and $d \in D$, $\delta(n) = d$ iff $\ell(n) \in \Sigma \times \{d\}$.
- Further, there exists a mapping $f : \mathbb{D} \to \mathbb{D}$ which maps, for any $c \in C_D$, data values occurring in nodes with tag $\Sigma \times \{c\}$ to distinct data values in $[c]$. This relies on the fact that our encoded formula forbids the same data value to occur in nodes with distinct comparison tags. Moreover, by density of the order, we can take this mapping to be injective. Applying this data renaming, we obtain a model of $(\varphi)^D$ of the form t^D, hence a model t of φ.

It is now clear why we could not add data tests in NonMixingXPath: it is not data-insensitive as CoreXPath fragments, but does not allow the (crucially mixed) axiomatization that was needed to obtain equisatisfiability for data-sensitive fragments. Note that this impossibility is slightly mitigated by the fact that some data tests are natively available in NonMixingXPath (in the form $\pi_{\triangle} \triangle d$).

8.3. Advanced Extensions

8.3.1. $\pi \triangle \pi$: Positive Data Joins. We observe that most of our fragments can be extended to allow restricted occurrences of data joins. Intuitively, we allow data joins in positions that guarantee that the join will be evaluated only once during satisfaction checking, which allows us to replace it by two tests against a specially chosen data constant. Hence, we extend any fragment with

$$\begin{aligned}
\pi_+ ::= {} & \pi \mid /\pi_+ \mid \pi_+/\pi_+ \mid \pi_+[\varphi_+] \mid \pi_+ \texttt{ union } \pi_+ \mid \pi_+ \texttt{ except } \pi \\
& \mid \pi_+ \texttt{ intersect } \pi_+ \mid \texttt{some \$x in } \pi_+ \texttt{ satisfies } \varphi_+ \\
\varphi_+ ::= {} & \varphi \mid \pi_+ \mid \varphi_+ \texttt{ or } \varphi_+ \mid \varphi_+ \texttt{ and } \varphi_+ \\
& \mid \pi_+ \texttt{ is } \pi_+ \mid \pi_+ \triangle \pi_+ \mid \pi_+ \triangle d
\end{aligned}$$

Productions for constructs such as navigation to the root, intersection, node comparison, etc. should only be considered in fragments where they are allowed. Note that we explicitly include in the extension several constructs like intersection that could be defined as syntactic sugar, because treating them as such would result in fewer allowed data joins. We justify this extension for all relevant fragments: NonMixingXPath does not support the mixing data tests required in our encoding, and PositiveXPath already supports positive data joins in its baseline version.

Proposition 8.8. *Positive joins are encodable in CoreXPath 1.0, CoreXPath 2.0 and EMSO^2-XPath.*

To prove this result, we fix below an ambient fragment among CoreXPath 1.0, CoreXPath 2.0 and EMSO^{2}XPath. We consider formulas φ_+ of the fragment, extended with positive data joins, and where joins are decorated with distinct marks in $\mathcal{M}$. We shall encode such formulas to φ formulas in the fragment without data joins but with data tests against constants, which we have shown to be admissible. To justify this extension, we design a translation which associates to any φ_+ expression an equisatisfiable φ expression.

The translation actually works over *marked* φ_+ expressions. Given a query φ_+, we can annotate each occurrence of a data join with a unique mark m from a finite set $\mathcal{M}$. For example,

$$c[@a \text{ eq } @b[. \text{ ne } \texttt{preceding::}*/@b]]$$

might be annotated using $\mathcal{M} = \{m, n\}$ as

$$\psi = c[@a \text{ eq}_m @b[. \text{ ne}_n \texttt{preceding::}*/@b]]$$

Then, given a valuation $\alpha\colon \mathcal{M} \to \mathbb{D}$, we define the translation $(\varphi_+)^\alpha$ as follows, showing only the key cases:

$$\begin{aligned}(\pi)^\alpha &= \pi \\ (\pi_+ \text{ eq}_m \pi'_+)^\alpha &= \pi_+ \text{ eq } \alpha(m) \text{ and } \pi'_+ \text{ eq } \alpha(m) \\ (\pi_+ \text{ ne}_m \pi'_+)^\alpha &= \pi_+ \text{ eq } \alpha(m) \text{ and } \pi'_+ \text{ ne } \alpha(m)\end{aligned}$$

Continuing the previous example with $\alpha(m) = d$ and $\alpha(n) = d'$, we have:

$$\begin{aligned}(\psi)^\alpha = c[&d \text{ eq } @a \text{ and} \\ &d \text{ eq } @b[. \text{ eq } d' \text{ and } d' \text{ ne } \texttt{preceding::}*/@b]]\end{aligned}$$

Note that this formula is satisfiable iff $d = d'$.

Obviously, $t, p \models (\varphi_+)^\alpha$ implies $t, p \models \varphi_+$ for any α. Conversely, if $t, p \models \varphi_+$, we show that there exists α such that $t, p \models (\varphi_+)^\alpha$. Roughly, α is chosen to assign to each m the data value that made the corresponding join pass. We conclude the argument by observing that there are only finitely (but exponentially) many $(\varphi_+)^\alpha$ up to satisfiability, hence $\texttt{or}_\alpha(\varphi_+)^\alpha$ is well-defined and equisatisfiable with φ_+. We give more details after the next paragraph.

As for variables, the encoding proposed here is not polynomial, but the added feature does not bring any complexity jump. We can actually use fragment-specific encodings that avoid the explicit constants of $(\varphi)^\alpha$: this can be achieved either by using second-order variables when available or similarly, in CoreXPath 1.0, thanks to the ability to have multiple propositional variables satisfied at a node in the decision procedure.

LEMMA 8.9. *For any t, p, φ_+ and α we have that $t, p \models (\varphi_+)^\alpha$ implies $t, p \models \varphi_+$ (and similarly for path expressions).*

PROOF. This follows easily by induction on φ_+. Consider for instance the case where $\varphi_+ = \pi_+ \text{ ne}_m \pi'_+$, we have $t, p, q \models (\pi_+)^\alpha$ and $t, p, q' \models (\pi'_+)^\alpha$ with $\delta(q) \sim \alpha(m)$ and $\alpha(m) \not\sim \delta(q')$. By induction hypothesis we have we have $t, p, q \models \pi_+$ and $t, p, q' \models \pi'_+$, and the data has not changed, which allows us to conclude. By construction of the φ_+ fragment, data joins can only occur under "positive" constructs, hence all other cases go well. For instance, consider (in fragment where it is relevant) the case where $\varphi_+ = \pi_+ \texttt{ except } \pi$. We have $t, p, q \models (\pi_+)^\alpha$ for some q for which $t, p, q \not\models \pi$. By induction hypothesis, we obtain $t, p, q \models \pi_+$ which allows us to conclude. □

We did not use the unicity of marks in φ_+. This comes into play in the next lemma.

LEMMA 8.10. *For any t, p and φ_+ such that $t, p \models \varphi_+$, there exists α over the marks of φ_+ such that $t, p \models (\varphi_+)^\alpha$ (and similarly for path expressions).*

PROOF. We proceed again by induction on expressions. Consider the case where $\varphi_+ = \pi^1_+ \triangle_m \pi^2_+$. We have $t, p, q_1 \models \pi^1_+$ and $t, p, q_2 \models \pi^2_+$, with $\delta(q_1)$ and $\delta(q_2)$ related according to $\triangle$. By induction hypotheses we obtain $t, p, q_1 \models (\pi^1_+)^{\alpha_1}$ and $t, p, q_2 \models (\pi^2_+)^{\alpha_2}$. Moreover, α_1 and α_2 have disjoint domains. We set $\alpha = \alpha_1 \uplus \alpha_2 \uplus \{m \mapsto \delta(q_1)\}$ and conclude easily that $t, p \models (\varphi_+)^\alpha$. Other cases are similar. □

LEMMA 8.11. *Let φ_+ a marked formula, and let M its set of marks. Given two valuations α and β in $M \to D_M \cup D(\varphi_+)$, we define $\alpha \approx \beta$ by*

(1) for all $m, m' \in M$, $\alpha(m) \sim \alpha(m')$ iff $\beta(m) \sim \beta(m')$, and
(2) for all $m \in M$, $d \in D(\varphi_+)$, $\alpha(m) \sim d$ iff $\beta(m) \sim d$.

Then, $(\varphi_+)^\alpha$ and $(\varphi_+)^\beta$ are equisatisfiable whenever $\alpha \approx \beta$.

PROOF. Assume that $\alpha \approx \beta$. If $(\varphi_+)^\alpha$ has a model t_α, replacing every occurrence of the datum $\alpha(m)$ by $\beta(m)$ for every mark m produces a model t_β of $(\varphi_+)^\beta$. The tree structure of t_α and t_β are the same, and for every data tests against constants from $(\varphi_+)^\alpha$ that hold somewhere in t_α, the corresponding test in $(\varphi_+)^\beta$ holds at exactly the same positions in t_β. □

We are now ready to prove Proposition 8.8.

PROOF OF PROPOSITION 8.8. Given an initial query φ_+ in any of these fragments, there are infinitely many $(\varphi_+)^\alpha$, but we shall see that only finitely many of these formulas is enough for our purpose.

Let $D(\varphi_+)$ be the data values occurring in data tests against constants in φ_+. Let M be the finite set of marks that occur in φ_+, and let D_M be a subset of $\mathbb{D}$ of cardinal $|M|$ and disjoint from $D(\varphi_+)$. We claim that if φ_+ is satisfiable, then there is α with images in $D_M \cup D(\varphi_+)$ such that $(\varphi_+)^\alpha$ is satisfiable. Starting with a model t' of φ_+, it suffices to take α' provided by the previous proposition, transform the model t' into t by rename values in the image of α' and outside the desired range, to obtain a suitable t and α.

Since the relation $\approx$ from Lemma 8.11 has only finitely many equivalence classes, the formula $\mathtt{or}_\alpha(\varphi_+)^\alpha$ is well-defined and equisatisfiable with φ_+. □

8.3.2. `last()`: Positional Predicates. The typical use of `last()` in XPath is through a *positional predicate* $\pi[\mathtt{position()} = \mathtt{last()}]$ or $\pi[\mathtt{last()}]$ that only keeps the last node in the document order among all those selected by π. We can also check whether a node is the ith one for some $i > 0$ with $\pi[i]$, or not the last one with $\pi[\mathtt{position()} \mathrel{!}= \mathtt{last()}]$ or not the ith one with $\pi[\mathtt{position()} \mathrel{!}= i]$. As seen in Table 8.1, these constructions are quite frequent in the benchmark. Here we discuss the case of `last()` and its negation, but the other positional predicates can be handled in a similar fashion.

As explained in §6.2.5.2, this kind of predicates is not supported in our simplified semantics, thus we rather focus on the one-argument functions $\mathtt{last}(\pi)$ and *notlast*(π)—the latter is not a standard function—, with semantics $[\![\mathtt{last}]\!]_{\mathcal{F}}(S) = \max_{\ll} S$ and $[\![\textit{notlast}]\!]_{\mathcal{F}}(S) = S \setminus \{\max_{\ll} S\}$ for any $S \subseteq N$.

Recall from §6.2.5.2 that `last()` can be expressed natively in CoreXPath 2.0 and thus in NonMixingXPath. The question here is to which extent it can be handled in the other fragments.

8.3.2.1. *Negative Results.* Our first result is that in some cases, the `last()` and *notlast*() functions cannot be expressed.

PROPOSITION 8.12. *The following path is not expressible in VerticalXPath.*

$$\texttt{last}(\texttt{ancestor}::a)[\texttt{child}::b] \tag{8.6}$$

If this query could be expressed by a formula φ in VerticalXPath, φ could be assumed not to contain any data test, since the evaluation of the query (8.6) on a data tree does not depend on the tree's data.

Hence, to study this problem, we can forget about the data in our models, and we will look at a small fragment of CoreXPath 1.0 containing only the vertical axes, noted VerticalCoreXPath and defined by the abstract syntax

$$\begin{aligned}
\alpha &::= \texttt{child} \mid \texttt{descendant} \mid \texttt{parent} \mid \texttt{ancestor} \\
\pi &::= \alpha::* \mid \pi/\pi \mid .[\varphi] \\
\varphi &::= \pi \mid a \mid \texttt{false}() \mid \texttt{not}(\varphi) \mid \varphi \texttt{ or } \varphi
\end{aligned}$$

To study whether the query (8.6) is expressible in this fragment, we will define and use data-free *bisimulations*. This is a completely standard approach for modal logics [Blackburn et al., 2001] tailored here for VerticalCoreXPath. These bisimulations can be seen as a variant of Ehrenfeucht-Fraïssé games for modal logics. Let ℓ, ℓ' be two labelled trees $N \to \Sigma$ (or XML documents for which we will ignore the data), let p (resp. p') be a node from ℓ (resp. ℓ'). Two players take part in the game, called Spoiler and Duplicator. The game starts with a pebble on p and a pebble on p'. If those two nodes are not labelled by the same letter, Spoiler wins the game. A game's step goes as follows:

(1) Spoiler moves one of the pebbles according to an axis α among `child`, `parent`, `descendant`, `ancestor`.
(2) Duplicator must move the other pebble according to the same axis, and on a node labelled by the same letter than the node chosen by Spoiler. If he cannot make such a move, Spoiler wins.

This game corresponds to the following bisimulation.

DEFINITION 8.13. Let ℓ and ℓ' be two labelled trees of domains N and N'. Let $Z \subseteq N \times N'$. Z is a *bisimulation* if, for all $p \in N, p' \in N'$ if $p\ Z\ p'$, then:

Harmony: p and p' have the same label,
Zig: for all $p \star y$, there exists $p' \star y'$ such that $y\ Z\ y'$ (for $\star \in \{\downarrow, \downarrow^{-1}, \downarrow^{+}, (\downarrow^{-1})^{+}\}$), and
Zag: for all $p' \star y'$, there exists $p \star y$ such that $y\ Z\ y'$ (for $\star \in \{\downarrow, \downarrow^{-1}, \downarrow^{+}, (\downarrow^{-1})^{+}\}$)

Then, the following result links bisimulations and logical equivalence.

LEMMA 8.14. *Let ℓ and ℓ' be two labelled trees, and let p (resp. p') be a node from ℓ (resp. ℓ'). If the nodes p and p' are bisimilar, then p and p' are logically equivalent for VerticalCoreXPath.*

The version of the game restricted to n rounds corresponds to the following definition of n-bisimulation.

DEFINITION 8.15. Let ℓ and ℓ' be two labelled trees. Let $(Z_i)_{i\leq n}$ be a sequence of relations between N and N'. For all j, $(Z_i)_{i\leq j}$ is a *j-bisimulation* if, for all $p \in N, p' \in N'$ if $p\ Z_j\ p'$, then

Harmony: p and p' have the same label,
Zig: for all $p\star y$, there exists $p'\star y'$ such that $y\ Z_{j-1}\ y'$ and $(Z_i)_{i\leq j-1}$ is a $(j-1)$-bisimulation (for $\star \in \{\downarrow, \downarrow^{-1}, \downarrow^+, (\downarrow^{-1})^+\}$), and
Zag: for all $p'\star y'$, there exists $p\star y$ such that $y\ Z_{j-1}\ y'$ and $(Z_i)_{i\leq j-1}$ is a $(j-1)$-bisimulation (for $\star \in \{\downarrow, \downarrow^{-1}, \downarrow^+, (\downarrow^{-1})^+\}$)

In order to get a result linking n-bisimulation and logical equivalence, we must first define the set of formulas using at most n navigational steps.

DEFINITION 8.16. We define a function *ns* by induction on the formulas of VerticalCoreXPath:

$$\begin{aligned}
ns(a) &= 0 \\
ns(\alpha::*) &= 1 \qquad \text{(where } \alpha \in \{\texttt{parent}, \texttt{child}, \texttt{ancestor}, \texttt{descendant}\}\text{)} \\
ns(\varphi \text{ or } \varphi') &= ns(\varphi \text{ or } \varphi') = \max\{ns(\varphi), ns(\varphi')\} \\
ns(\texttt{not}(\varphi)) &= ns(\varphi) \\
ns(.[\varphi]) &= ns(\varphi) \\
ns(\pi/\pi') &= ns(\pi) + ns(\pi')
\end{aligned}$$

If $ns(\varphi) = n$, we say that φ has n *nested steps*. We note $\mathsf{VerticalCoreXPath}^n = \{\varphi \in \mathsf{VerticalCoreXPath} \mid ns(\varphi) \leq n\}$.

DEFINITION 8.17. Let ℓ, ℓ' be two labelled trees, and let p (resp. p') be a node of ℓ (resp. ℓ'). We note $p \equiv_n p'$ if p and p' are *logically equivalent* for $\mathsf{VerticalCoreXPath}^n$.

LEMMA 8.18. *Let $n \geq 0$. Let ℓ and ℓ' be two labelled trees, let p (resp. p') be a node of ℓ (resp. ℓ'), and let $(Z_i)_{i\leq n}$ be an n-bisimulation between ℓ and ℓ'. If $p\ Z_j\ p'$ for some $j \leq n$, then $p \equiv_j p'$.*

PROOF. We prove this lemma by induction on j:

- $j = 0$: the only formulas are label tests, and 0-bisimulation between two nodes require that the nodes have the same label. So 0-bisimilar nodes are logically equivalent for formulas of 0 nested steps.
- Let us assume that $p\ Z_j\ p'$ and that the theorem is true for i-bisimulations with $i < j$. First, since $p\ Z_j\ p'$ implies $p\ Z_{j-1}\ p'$, then $p \equiv_{j-1} p'$ by induction hypothesis. Now, let us consider a formula $\varphi = \alpha::*/\varphi'$ with $\star$ being the relation corresponding to α. If φ is true on p, then there exists $p \star y$ such that φ' is true at y. Then we choose $p' \star y'$ such that $y\ Z_{j-1}\ y'$ (such a node exists because $(Z_i)_{i\leq j}$ is a j-bisimulation). And now, because $ns(\varphi') = j - 1$, by induction hypothesis, φ' is true at y', so φ is true at p'. Symmetrically, we can prove that if φ is true at p', then it is also true at p. Now for the case $\varphi = .[\varphi']$ we study φ' instead, and the Boolean combinations are handled easily. At the end, we have $p \equiv_j p'$. □

Now, let us assume that the query (8.6) could be expressed by a formula φ in VerticalCoreXPath. Let $n = ns(\varphi)$. We show in Appendix A.1.1 that there exist a tree ℓ_n and two nodes

$p, p' \in N_n$ such that p and p' are n-bisimilar, but such that p satisfies the query (8.6), and p' does not.

In the end, this result shows that $\texttt{last}()$ cannot be expressed in VerticalXPath nor in DownwardXPath, even for simple one-step paths. Furthermore, we can show that it cannot be polynomially encoded in DownwardXPath by adapting the hardness proofs of [Figueira and Segoufin, 2009].

PROPOSITION 8.19. *Satisfiability in DownwardXPath extended with both the path* $\texttt{last}(\texttt{descendant-or-self::}*)$ *and the path* $\mathit{notlast}(\texttt{descendant-or-self::}*)$ *is* ACKERMANN*-hard.*

PROOF. We adapt the proof of Figueira and Segoufin [2009, Cor. 1]. Our aim is to build our formula so as to ensure that the entire simulation of the incrementing counter machine is performed along a single branch of the tree—this will be the rightmost branch. We pick a fresh label LAST (which is also used in [Figueira and Segoufin, 2009, Thm. 2]) and first require

$$\texttt{last}(\texttt{descendant-or-self::}*)[\textsc{last}] \tag{8.7}$$

so that the last leaf of the (sub)tree below the point of evaluation, in the document order, is labelled by LAST. We then make sure that no other node in the (sub)tree is not labelled by LAST

$$\texttt{not}(\mathit{notlast}(\texttt{descendant-or-self::}*)[\textsc{last}]) \tag{8.8}$$

We then apply the construction of [Figueira and Segoufin, 2009, Prop. 1], but whenever a step $\texttt{descendant::}*$ would be used, we replace it by $\texttt{descendant::}*[\texttt{descendant-or-self::}\textsc{last}]$ to ensure that we only move along the rightmost branch. □

8.3.2.2. *Simple Uses of* $\texttt{last}()$. However, we can still look for some uses of $\texttt{last}()$ that can be reasonably allowed. In particular, the path expressions from the statement of Proposition 8.19 can be handled in ForwardXPath—or at least in its *regular* extension [ten Cate, 2006] with new axes and the Kleene plus and Kleene star operators on paths

$$\alpha ::= \cdots \mid \texttt{previous}-\texttt{sibling} \mid \texttt{next}-\texttt{sibling}$$
$$\pi ::= \cdots \mid \pi^+ \mid \pi^*$$

Their semantics are defined by $[\![\texttt{previous}-\texttt{sibling}]\!]_{\mathsf{A}} = \rightarrow^{-1}$, $[\![\texttt{next}-\texttt{sibling}]\!]_{\mathsf{A}} = \rightarrow$, $[\![\pi^+]\!]^{\nu}_{\mathsf{P}} = ([\![\pi]\!]^{\nu}_{\mathsf{P}})^+$, $[\![\pi^*]\!]^{\nu}_{\mathsf{P}} = ([\![\pi]\!]^{\nu}_{\mathsf{P}})^*$. As far as the computational complexity of satisfiability is concerned, this extension comes 'for free' in CoreXPath 1.0 [Afanasiev et al., 2005; Marx, 2005], ForwardXPath [Figueira, 2012b, Thm.6.4] (with $\texttt{next-sibling}$ but without $\texttt{previous-sibling}$), VerticalXPath [Figueira and Segoufin, 2017, Thm. 2.1], and DownwardXPath [Figueira, 2012a, Thm. 6.4] (without the new axes). We show in Appendix A.1.2 that we can handle using this regular extension $\texttt{last}(\pi)$ and $\mathit{notlast}(\pi)$ on any *one-step* path argument of the form $\pi = \alpha::*[\varphi]$. They can even be expressed in ConditionalXPath [Marx, 2005]. However, this sometimes requires axes that not available in some of the fragments; for instance

$$\begin{aligned}\texttt{last}(\texttt{descendant::}*[\varphi]) \equiv \big(\texttt{child::}*\big[&\texttt{descendant-or-self::}*[\varphi] \text{ and}\\ &\texttt{not}(\texttt{following-sibling::}*/\texttt{descendant-or-self::}*[\varphi])\\ \big]\big)^+[&\varphi \text{ and } \texttt{not}(\texttt{descendant::}*[\varphi])]\end{aligned}$$

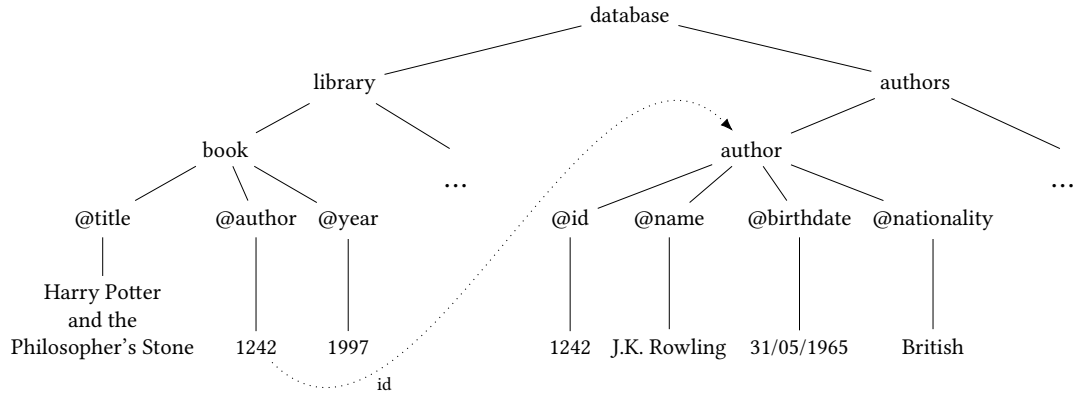

FIGURE 8.1. Example of an `id` jump.

Since this translation require the use of `following-sibling::*`, we cannot allow it in the regular extensions of VerticalXPath and DownwardXPath, but it can be allowed in the regular extension of ForwardXPath.

We can also handle *two-step* paths $\alpha_1::*[\varphi_1]/\alpha_2::*[\varphi_2]$ in RegularXPath using similar ideas, but the translation becomes rather complicated in some cases. Moreover, there is no occurrence of the use of `last()` on a path of length three or more in our benchmark, and only a few queries are using `last()` on a two-step path. For this reason, we did not further investigate the possible translation of more complex paths, and we only handle `last()` and *notlast*() on one-step paths in our benchmark (for fragments that cannot use the encoding given in (6.5)).

8.3.3. `id()`: Jumps. Another interesting XPath feature in the XML document model is the id/idref mechanism [Benedikt and Koch, 2009; Marx, 2003]. In full generality, this mechanism relies on attributes declared as identifiers in the XML schema, but we shall consider a simpler setting where the `@id` attribute plays this role. The function `id()` then takes a path as argument, and returns the nodes of the document (if any) that have an `@id` attribute matching the datum found at the end of the path given as argument.

$$[\![\texttt{id}]\!]_{\mathcal{F}}(S) = \{p \mid \exists p' \in S \,.\, \delta(p') = \delta([\![\texttt{@id}]\!]^{\nu}_{\mathrm{P}}(p))\}$$

The function `idref()` is its inverse. For instance, in Figure 6.1, evaluating `id(@ref)` at the t node returns the o node. This feature allows to do complex and precise jumps from one node to another inside the document.

EXAMPLE 8.20. For instance, on the data tree from Figure 8.1, the query

`id(database/library/book[@title = ‘‘HarryPotter...”]/@author)/@name`

will jump as shown by the dotted arrow and return “J.K. Rowling”.

8.3.3.1. *Undecidability.* Adding `id()` jumps makes most of the fragments undecidable, as soon as we can also use (full) data joins or node tests π `is` π'. The proof in Appendix A.2 reduces from Post's correspondence problem.

PROPOSITION 8.21. *Satisfiability is undecidable in both DownwardXPath and CoreXPath 2.0 extended with* `id()`.

TABLE 8.2. Number of new queries captured by the extensions of Section 8.2 in each fragment.

	basic	$\pi \triangle \pi$	last()	id()
Positive	7,653			
Core 1.0	7,895	+243	+54	
Core 2.0	4,309	+266		
Downward	1,993		+12	
Vertical	7,974		+25	+0
Forward	2,053		+26	
EMSO^2	4,760	+241		+11
NonMixing	136			

8.3.3.2. *Root-level* id(). Although we cannot allow arbitrary uses of id(), there might be some practically relevant ways of handling it. For fragments that allow them, variables and data joins seem the be a good way to encode the target set of an id() jump, but we must be able to axiomatise these variables to contain exactly the target nodes. Such an axiomatisation is possible if we can identify exactly the initial node from which the query has been evaluated. This can be done if id() appears in a root-level path, in a fragment endowed with data joins and variables (see Appendix A.2 for details).

$$\pi ::= \cdots \mid /\pi_{id}$$
$$\pi_{id} ::= \pi \mid \pi_{id}/\pi_{id} \mid \pi_{id} \texttt{ union } \pi_{id} \mid \pi_{id}[\varphi] \mid \texttt{id}(\pi_{id})$$

We allow these additions for the only two fragments able to handle simultaneously variables and data joins: VerticalXPath and EMSO^{2}XPath.

Moreover, in EMSO^{2}XPath we can allow id() jumps not only in root-level paths but also in top-level paths:

$$\pi' ::= \pi \mid \pi_{id}$$

Indeed, we can use a second-order variable in the axiomatisation to keep track of the starting node in the query evaluation (see Appendix A.2 for details).

8.4. Extended Benchmark Results

We have implemented the extensions of this section as Relax NG schemas. The results on the benchmark are presented in violet in Figures 8.2 and 8.3. Generally, the differences observed before still hold but are significantly lessened. Strikingly, the extensions even bring CoreXPath 1.0 above NonMixingXPath: the latter only supports non-mixing data tests. It also differentiates VerticalXPath from the two other DataXPath fragments, due to its support for root paths, which in turn allows to support free variables.

Looking at the influence of each extension separately, Table 8.2 shows that the 'basic' extensions of Sections 8.2.1 to 8.2.3 contribute most of the gains, while adding positive joins, positional predicates, or id() jumps to the basic extensions only brings small improvements. Regarding positional predicates, we found that many occurrences are of the form last($x), which is outside the scope of our treatment in Section 8.3.2. Regarding id(), Table 8.1 shows that there are *very* few occurrences of id() in the benchmark; one of these examples is shown

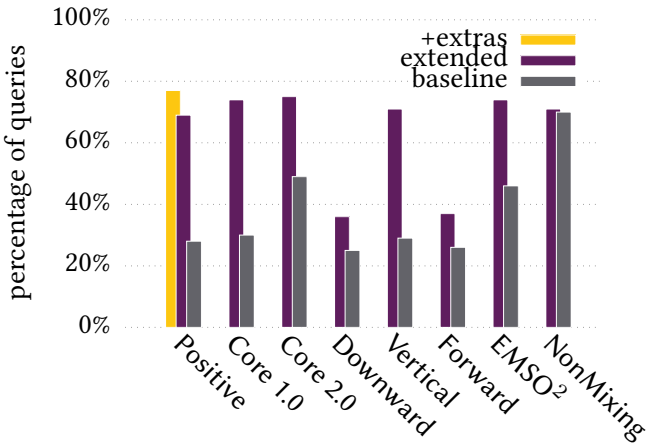

FIGURE 8.2. Coverage of the XSLT sources.

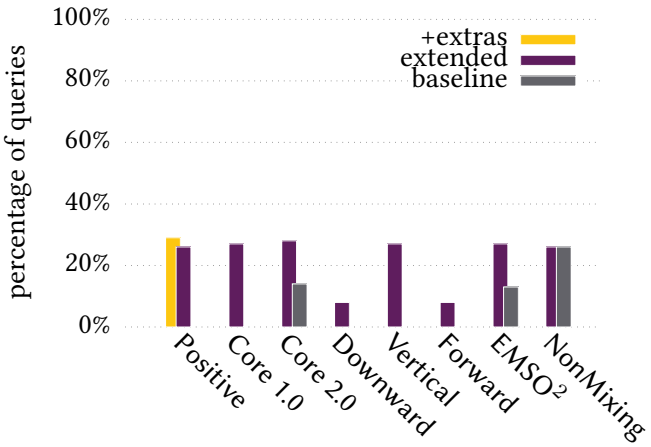

FIGURE 8.3. Coverage of the XQuery sources.

in Example 6.4. A quick investigation of the usage of the `id` attribute in the benchmark shows that developers rather interact with it through variables and data tests.

8.5. Discussion

Figures 8.2 and 8.3 show the coverage of the benchmark: in grey for the baseline fragments from Figures 7.2 and 7.3 and in violet for their extensions described in Chapter 8; the yellow 'extras' are the topic of §8.5.2.2. The combined coverage of the extended fragments on the full benchmark is of 12,867 queries (60.86%).

Leaving DownwardXPath and ForwardXPath aside, the extended versions of the remaining six fragments have a somewhat similar coverage: for XSLT queries, between 69.16% for PositiveXPath and 75.03% for CoreXPath 2.0, and for XQuery ones, between 26.35% for NonMixingXPath and 28.08% for CoreXPath 2.0. We look more closely at the differences between the fragments in Section 8.5.1.

TABLE 8.3. Combined coverage of the extended fragments by query size.

sizes	1–4	5–8	9–12	13–16	17–20	21–24	25–28	29–32	33–36	37–40	41–44	45–48	≥ 49
queries	5,146	5,661	3,359	1,996	1,330	806	590	503	283	231	117	131	988
coverage	97.9%	58.7%	42.0%	44.2%	54.8%	44.9%	53.0%	42.7%	40.9%	47.6%	35.8%	46.5%	25.7%

TABLE 8.4. Difference matrix for the extended fragments. Cell (i, j) shows the number of queries covered by fragment i but not fragment j.

	Positive	Core 1.0	Core 2.0	Downward	Vertical	Forward	EMSO^2	NonMixing
Positive	0	4	0	6,283	300	6,159	23	331
Core 1.0	889	0	0	6,917	469	6,765	124	681
Core 2.0	942	57	0	6,974	526	6,822	181	689
Downward	251	0	0	0	0	0	12	83
Vertical	746	30	30	6,478	0	6,478	73	671
Forward	279	0	0	152	152	0	26	85
EMSO^2	796	12	12	6,817	400	6,679	0	639
NonMixing	584	49	0	6,368	478	6,218	119	0

Obviously, the coverage of XPath queries extracted from XQuery files is quite poor compared to that of XSLT files. Among the other factors contributing to the coverage, we see that the size of the query is (negatively) correlated (Table 8.3). Another correlation is the presence of at least one axis step, where the combined coverage is of 74.50%, but only 48.79% for queries without any axis step. The main factor we identify is however the presence of non-standard or unsupported function calls in the query, which we discuss in Section 8.5.2.

8.5.1. Comparisons Between Fragments. In the case of the extended fragments of Chapter 8, the inclusions of Figure 7.1 are slightly changed: CoreXPath 2.0 now contains NonMixingXPath and PositiveXPath is included into CoreXPath 2.0 but disjoint from NonMixingXPath.

These theoretical inclusions are reflected in the difference matrix shown in Table 8.4 and the accompanying chord graph of Figure 8.4. On this graph, a chord between fragments i and j has thickness proportional to entry (i, j) on its i end, and to entry (j, i) on its j end; and the colour of the chord is the one of the 'winning' value. An interactive version is available online at `http://www.lsv.fr/~schmitz/xpparser/`, where clicking on a chord will provide examples extracted from the benchmark. There are three maximal incomparable fragments, namely CoreXPath 2.0, VerticalXPath, and EMSO^{2}XPath. Among the fragments of DataXPath, VerticalXPath benefits from the support of most extensions, while—as seen in Table 6.2—horizontal navigation is not used very frequently in the benchmark. The coverage of the extended CoreXPath 2.0 is almost as large as the combined coverage: only 30 queries from VerticalXPath are not captured by CoreXPath 2.0, and they all contain data joins under a negation; only 12 queries from EMSO^{2}XPath are not captured, which include 11 queries with `id()` plus one of the previous 30.

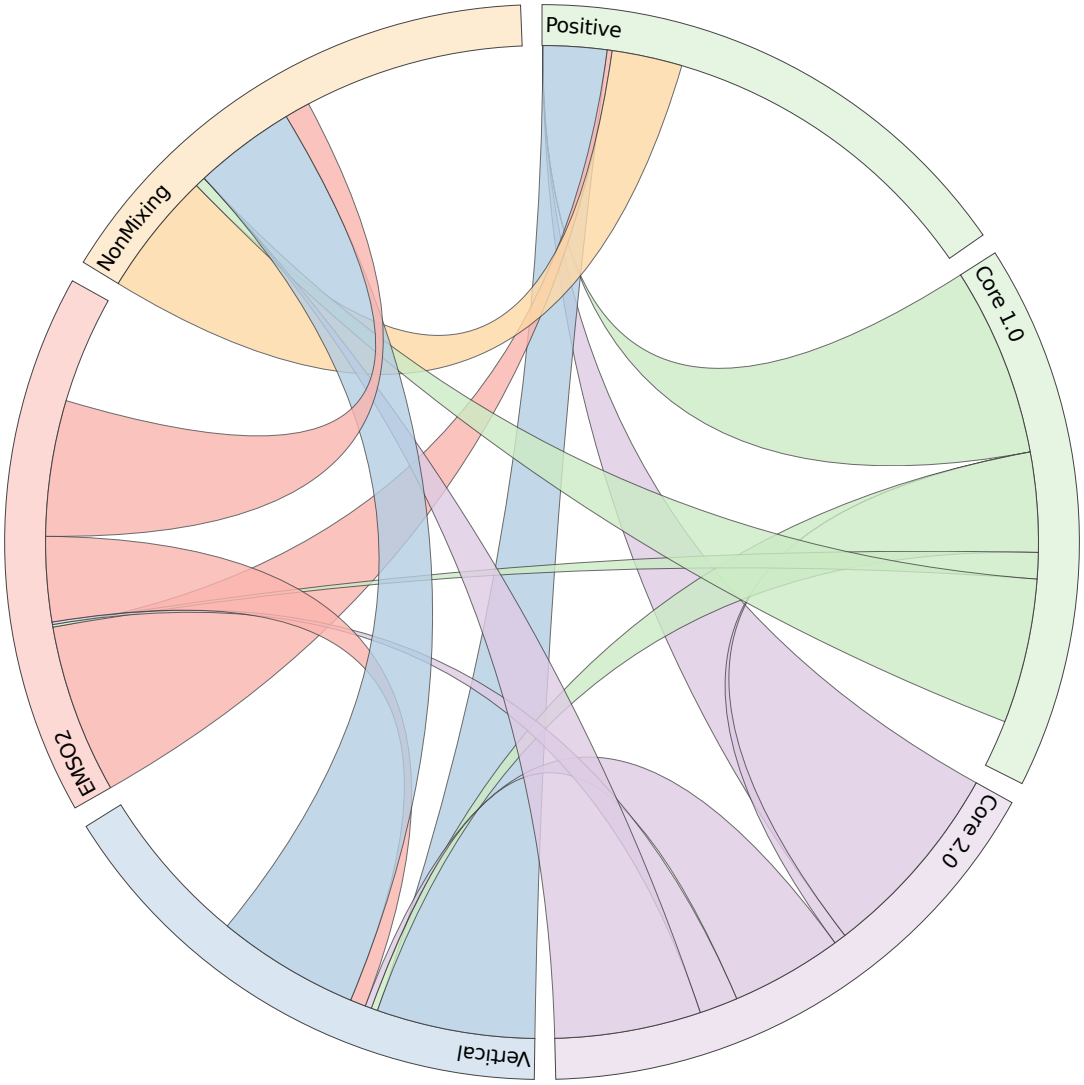

FIGURE 8.4. Chord graph of the difference matrix of Table 8.4, without Downward and ForwardXPath.

From a more practical perspective, we think that the extended versions of PositiveXPath and CoreXPath 1.0 are the most promising ones: satisfiability has a more manageable complexity (NP-complete and EXP-complete, resp.), and the coverage is not too far behind CoreXPath 2.0 (with 942 and 57 fewer queries, resp.). Note that PositiveXPath is nearly included into CoreXPath 1.0, with only four queries (featuring intersections) not captured by CoreXPath 1.0.

8.5.2. Supporting Functions. Due to the large number of calls to non-standard functions in the benchmark, the coverage of 'XPath 3.0 std' (cf. Table 6.1) is an upper bound on the achievable coverage. With respect to the number of queries captured by 'XPath 3.0 std',

the combined coverage is 78.33%, and the precise coverage varies between 75.71% for PositiveXPath and 82.14% for CoreXPath 2.0 in XSLT sources and between 56.30% for NonMixingXPath and 60.00% for CoreXPath 2.0 in XQuery sources: the latter sources are more complex even when leaving aside their higher reliance on non-standard functions.

A remaining issue is the support of standard functions. For instance, the four functions that occur the most frequently in the benchmark are in decreasing order `count()`, `concat()`, `local-name()`, and `contains()`, and they are all standard; `local-name()` is supported in our fragments, but the remaining three are not.

8.5.2.1. *Aggregation.* CoreXPath 1.0 extended with node expressions $\texttt{count}(\pi) \mathrel{\triangle^+} i$ for an integer i can be translated into the two-variable fragment of first-order logic with counting on trees, which has an EXPSPACE decision procedure [Bednarczyk et al., 2017]. There are 314 occurrences of such expressions out of the 624 occurrences of `count()` in the benchmark, but unfortunately not a single query is gained by adding this feature to the extended CoreXPath 1.0 fragment. Capturing more occurrences of `count()` requires arithmetic operations, and leads to an undecidable fragment akin to AggXPath [Benedikt and Koch, 2009].

8.5.2.2. *String Processing and Arithmetic.* A promising direction for supporting more functions is the move to SMT solvers—even though it might also mean moving to semi-deciding satisfiability. Linear arithmetic is supported by all solvers, while theories comprising string concatenation, string length, and substring operations are also supported [e.g. Abdulla et al., 2015; Kiezun et al., 2013; Liang et al., 2016; Trinh et al., 2014; Zheng et al., 2013]. SMT solvers have already been used in [Benedikt and Cheney, 2010] to check XQuery inputs, using the classical interval encoding of trees, and the approach could be enriched to cover basic arithmetic and string support. Furthermore, a custom finite tree theory may be added to SMT solvers for more efficiency [e.g. Blackburn et al., 1996; Cheney, 2011].

These considerations lead us to adding support in the extended PositiveXPath for linear arithmetic, the standard functions `concat()`, `contains()`, `string-length()`, and similar ones such as `ends-with()`. We view this fragment as a good candidate for practical satisfiability checking. At 62.75%, the coverage of this fragment, shown in yellow in Figures 8.2 and 8.3, bests the combined coverage of our other fragments. This translates in particular to 84.77% of the subset of XSLT queries captured by 'XPath 3.0 std'.

This new fragment is incomparable with the others, with a new combined coverage of 67.40% (83.55% for XSLT sources). If we restrict our attention to the 14,732 queries (69.68%) that only use `not()` and the functions supported in this new fragment, the new combined coverage reaches 96.69% (98.10% for XSLT sources). Thus our fragments cover nearly all the queries that do not use unsupported or non-standard functions, from which we argue that improved function support is the most promising research avenue in order to gain over the extended fragments described in this chapter.

8.6. Concluding Remarks

We have designed a benchmarking infrastructure for testing the practical relevance of XPath fragments, based on the XQueryX format and Relax NG schemas. We have used this benchmark of over 20,000 queries, extracted from the XSLT and XQuery files of open-source projects, to evaluate the syntactic coverage of state-of-the-art XPath fragments for which decidability is known (or still open in the case of EMSO^{2}XPath). Concerning the benchmark

itself, it would of course be interesting to incorporate new sources, to confirm our observations on a larger scale.

Our analysis shows that, in a hypothetical satisfiability checker for XPath, the differences between the fragments defined in the theoretical literature are not as important as the differences introduced by the *front-end* translating real XPath inputs to the restricted syntax on which the decision procedure operates. Among the features that such a front-end should support, the most impactful ones would be free variables, data tests against constants (and constant expressions), positive joins, and positional predicates.

According to our benchmark results, such a front-end combined with the decidable fragments from the literature would cover about 70%–75% of the XPath queries found in XSLT files. However, due to the high reliance on user-defined or ill-supported functions, this drops to less than 30% for XPath queries from XQuery files: full-blown program analysis techniques seem necessary for XQuery.

As the support of XPath functions is a key factor, a promising approach might be to harness the power of modern SMT solvers to handle string-manipulating functions and linear arithmetic, which might cover 77.44% of the XPath queries from the XSLT sources.

CHAPTER 9

General Conclusion

Many logics on data words and data trees have been studied in the literature. These logics differ in expressivity, and in complexity: their satisfiability problems belong to various classes of complexity, and are often non-elementary or even undecidable. It is thus non-trivial to choose among these logics to study practical problems, since a trade-off must be made between expressivity and complexity. Furthermore, various techniques have been developed to study these logics, and they often lack modularity: it can be challenging to adapt the techniques developed for one logic to study another closely related logic.

This thesis had two objectives. On the one hand, we aimed at developing proof systems for data logics together with a proof strategy of optimal complexity for the validity problem, as we hoped such techniques would prove to be more modular. On the other hand, we wanted to measure to what extent various logics on data trees could capture practical uses of the XPath query language. We recall our contributions, and make some comments on our work.

Proof Systems. In Part 1, we investigated the design and use of proof systems as a mean to design effective algorithms solving the validity problem of some modal logics. Our main objective was to do so for a logic on data words, finite or infinite. However, one of the main selling points of proof systems is their modularity, and we started by designing a hypersequent calculus for a data free logic on linear structures: the tense logic $\mathbf{K_t4.3}$.

To do so, in Chapter 3 we improved the hypersequent calculus from Indrzejczak [2016] by adding clusters and annotations. This allowed our calculus to be better suited to represent weak total orders, thus taking advantage of the finite model property of $\mathbf{K_t4.3}$ for this class of models [Blackburn et al., 2001; Ono and Nakamura, 1980]. In the end, our hypersequent calculus is sound and complete for $\mathbf{K_t4.3}$, and enjoys a simple proof search strategy of optimal coNP complexity. Moreover, as for the calculus of Indrzejczak [2016], it can be easily enriched to handle some classical extensions of $\mathbf{K_t4.3}$ such as density or unboundedness while still having an optimal coNP proof search. This was the first evidence of the modularity of our approach.

Then, in Chapter 4, we turned to the study of the tense logic over ordinals $\mathbf{K_tL_\ell.3}$. Our main interest for well-founded structures was to capture ω-words, as they are used in many applications, but it turned out that adapting our calculus for any arbitrary ordinal was not any harder. Modifying one of our rules as done in Avron [1984] to deal with well-foundedness lead to a sound and complete proof system for $\mathbf{K_tL_\ell.3}$, as well as an optimal coNP proof search. Moreover, our work also lead to a small model property for $\mathbf{K_tL_\ell.3}$: it cannot express ordinals larger or equal to ω^2. Finally, $\mathbf{HK_tL_\ell.3}$ can be enriched to work over specific classes of ordinals, by testing validity only over models of order type smaller than α, for a given $\alpha < \omega^2$.

Finally, in Chapter 5, we enriched $\mathbf{K_tL_\ell.3}$ with freeze quantifiers à la Alur and Henzinger [1994], similarly to Demri and Lazić [2009], to work over data ordinals. Adding new rules to

our calculus to handle the freeze quantifiers lead to a sound hypersequent calculus ($\mathbf{HK^d_t L_\ell.3}$) for $\mathbf{K^\downarrow_t L_\ell.3}$. However, $\mathbf{K^\downarrow_t L_\ell.3}$ being undecidable [Figueira and Segoufin, 2009], we provided a decidable fragment, $\mathbf{K^d_t L_\ell.3}$, for which our calculus is complete and enjoys an optimal coNP proof search.

This fragment is closely related to the fragment of XPath with data tests and navigation among siblings XPath$(\leftarrow^+, \rightarrow^+, =)$ from Figueira and Segoufin [2009], the main difference being that $\mathbf{K^d_t L_\ell.3}$ cannot perform nested data tests. Figueira and Segoufin [2009] showed that XPath$(\leftarrow^+, \rightarrow^+, =)$ is undecidable, and that its restriction to the `following-sibling` axis only is ACKERMANN-complete. Hence, this restriction allowed us to get a logic of acceptable complexity.

Furthermore, all these logics are closely related to the two variable fragment of some first order logics: $\mathbf{K_t 4.3}$ is exactly as expressive as $\mathrm{FO}^2(<)$ [Etessami et al., 2002], and $\mathbf{K_t L_\ell.3}$ is equivalent to the same logic on ordinals. Similarly, we showed that $\mathbf{K^d_t L_\ell.3}$ is equivalent to the logic $\mathrm{FO}^2(<, \sim)$ from Bojańczyk et al. [2011]. For each case, the translation of a first order formula to the corresponding modal logic is exponential, thus giving an optimal coNEXP algorithm for validity in these logics.

In the end, the modularity of our hypersequent calculus allowed us to adapt it to various logics, and we believe that such an approach could be carried further. For instance, it would be interesting to see if our proof system is complete for $\mathbf{K^\downarrow_t L_\ell.3}$, and if an optimal proof search can be designed for decidable fragments of bigger complexity. Another interesting challenge would be the development of similar tools for logics over data trees.

XPath Benchmark. In Part 2, we investigated the practical impact of various theoretical works in the literature concerning the XPath query language. As the satisfiability problem of XPath is undecidable, many decidable fragments of XPath have been studied, and we designed a benchmark for testing the practical coverage of these theoretical fragments.

To do so, as described in Chapter 6, we extracted over 20,000 queries from various open-source projects, and designed schemas for testing the membership of a query to an XPath fragment. However, these theoretical fragments (presented in Chapter 7) were never meant to be used for real-world queries, and many of our queries did not belong to any of these baseline fragments. This lead to an interesting problem: which practical features of XPath can be added to a theoretical fragment without affecting its complexity?

To that end, in Chapter 8, we investigated six XPath features and determined to which fragments we could reasonably add them. We also provided some cases where an extension could not be handled by a fragment, for instance because adding it to the fragment would make it undecidable.

According to benchmark results, once enriched with our extensions, most of the fragments turned out to capture about the same amount of real-world queries, despite different complexities. Furthermore, when looking more closely at queries that are not captured by any fragment, the limiting factor seems to be the use of functions, user-defined or not. Hence, a possible future work would be to investigate the use of SMT solvers to support various XPath functions by some theoretical fragment, such as string processing functions or arithmetic functions. In that direction, a good candidate might be PositiveXPath.

From Words to Trees. A starting point for our work in Part 1 was a proof system developed by Baelde et al. [2016] for a PSPACE-complete logic on data trees, contained in Downward-XPath. Concerning this work, the natural next step would be the support of converse navigation. However, upward navigation inside a tree structure is trichotomic. Hence, working on linear structures comes with the same difficulties, and we believe our work could be carried further to develop proof systems on data trees, for instance towards supporting (fragments of) VerticalXPath.

Moreover, in both parts, we pointed out NP-complete fragments, which is the best complexity one can possibly hope for when studying the satisfiability problem. In particular, we think that PositiveXPath, one of the fragment that stood out from our benchmark, could be enriched by string-manipulating functions and linear arithmetic (see S8.5.2.2). This extension could be handled through oracle calls to an SMT solver, and such techniques have already been used alongside a proof system [Farooque and Graham-Lengrand, 2013; Rouhling et al., 2015], which also lead to an implementation [Graham-Lengrand, 2013]. Thus, it would be interesting to design an effective proof system for PositiveXPath and to investigate how it can be enriched with such oracle calls.

Technical Appendix

A.1. Expressiveness Results on last()

A.1.1. `last()` is not Expressible in the Vertical Fragment. We prove the following result, thanks to n-bisimulation techniques presented in §8.3.2.1.

Proposition 8.12. *The following path is not expressible in VerticalXPath.*

$$\texttt{last}(\texttt{ancestor}::a)[\texttt{child}::b] \tag{8.6}$$

Let us assume that the query (8.6) could be expressed by a formula φ in VerticalCoreXPath. Let $n = ns(\varphi)$. Let us show that there exist a tree ℓ_n and two nodes $p, p' \in N_n$ such that p and p' are n-bisimilar, but such that p satisfies the query (8.6), and p' does not.

The tree ℓ_n and the bisimulations are represented on Figures A.1 and A.2. It only have one branch, along which many a nodes appear. Half of them have a b child, and the other half are lones a, i.e. without a b child. Those two types of a nodes are alternating along the branch, and each of them are separated by a chain of c nodes of length $2n+1$: on Figures A.1 and A.2, a double edge represents a chain of c nodes of length n. Intuitively, this padding of c nodes is there to make the `child` and `parent` relations worthless for Spoiler if he only has n moves to work with. Then, we repeat this pattern enough times to ensure that Spoiler cannot win in n moves using the transitive relations, if the game starts at two distinct positions that are in the middle of the branch.

Formally, we define the tree ℓ_n as follows:

- We partition its set N_n of nodes into the subsets
  - $A_n = \{a_i \mid i \in [-2n+1, 2n-1]\}$
  - $B_n = \{b_{2i+1} \mid i \in [-n, n-1]\}$
  - $C_n^i = \{c_{i,k} \mid k \in [-n, n]\}$ for $i \in [-2n+1, 2n-2]$
- The relation $\downarrow$ holds between the following nodes:
  - $a_{2i+1} \downarrow b_{2i+1}$, for $i \in [-n, n-2]$
  - $c_{i,k} \downarrow c_{i,k-1}$ for $i \in [-2n+1, 2n-2]$ and $k \in [-n+1, n]$ (the $c_{i,k}$ are forming a chain of length $2n+1$)
  - $b_{2i+1} \downarrow c_{2i,n}$ and $c_{2i,-n} \downarrow a_{2i}$ for $i \in [-n+1, n-1]$ (the c chains link each b with the next lone a)
  - $a_{2i+2} \downarrow c_{2i+1,n}$ and $c_{2i+1,-n} \downarrow a_{2i+1}$ for $i \in [-n, n-2]$ (the c chains link each lone a with the next a)

We now describe the bisimulation relations $(Z_i)_{i \leq n}$ between the nodes of ℓ_n:

- $c_{-1,0}\ Z_n\ c_{0,0}$ (they are the starting nodes of our n rounds game)
- Duplicator must be able to simulate small moves of Spoiler locally around the starting nodes:

  - $\forall i \in [1, n],\ c_{-1,i}\ Z_{n-i}\ c_{0,i}$
  - $\forall i \in [1, n],\ c_{-1,-i}\ Z_{n-i}\ c_{0,-i}$
- Duplicator can simulate bigger moves from Spoiler by doing a shift:
  - $\forall i \in [0, n-2],\ a_{2i}\ Z_{n-1-i}\ a_{2i+2}$
  - $\forall i \in [0, n-2],\ a_{2i+1}\ Z_{n-2-i}\ a_{2i+3}$
  - $\forall i \in [-n+1, -1],\ a_{2i}\ Z_{n+i}\ a_{2i+2}$
  - $\forall i \in [-n, -1],\ a_{2i+1}\ Z_{n+i}\ a_{2i+3}$
  - $\forall i \in [0, n-2],\ c_{2i-1,k}\ Z_{n-1-i}\ c_{2i+1,k}$
  - $\forall i \in [0, n-2],\ c_{2i,k}\ Z_{n-1-i}\ c_{2i+2,k}$
  - $\forall i \in [-n+1, -1],\ c_{2i-1,k}\ Z_{n+i}\ c_{2i+1,k}$
  - $\forall i \in [-n+1, -1],\ c_{2i,k}\ Z_{n+i}\ c_{2i+2,k}$
- Duplicator can simulate too big moves from Spoiler by using the identity relation: $\forall p \in N_n,\ p\ Z_n\ p$
- Finally, every relation must contain the smaller ones: $\forall (p, p') \in N_n^2, \forall i < j$, if $p\ Z_j\ p'$ then $p\ Z_i\ p'$.

LEMMA A.1. *$(Z_i)_{i \leq n}$ is an n-bisimulation.*

PROOF. Let us prove by induction on j that $(Z_i)_{i \leq j}$ is a j-bisimulation.

- $j = 0$: every nodes in relation for Z_0 do have the same labels, so Z_0 is a 0-bisimulation.
- Let us assume that $(Z_i)_{i \leq j}$ is a j-bisimulation for some $j < n$, and let us prove that $(Z_i)_{i \leq j+1}$ is a $(j+1)$-bisimulation. Let p and p' be two nodes from ℓ_n such that $p\ Z_{j+1}\ p'$. Then:
  (1) By definition of Z_{j+1}, p and p' have the same label.
  (2) Let $\star \in \{\downarrow^{-1}, \downarrow, (\downarrow^{-1})^+, \downarrow^+\}$. We must prove that for every $p \star y$, there exists a $p' \star y'$ such that $y\ Z_j\ y'$. The cases $\star = \downarrow^{-1}$ and $\star = \downarrow$ are easy because we padded ℓ_n with long enough c-chains. Let us focus on the case $\star = (\downarrow^{-1})^+$ (the case $\star = \downarrow^+$ is similar): let $y \in N_n$ such that $p(\downarrow^{-1})^+ y$. Now, if $p'(\downarrow^{-1})^+ y$ then we can take $y' = y$ and use the identity relation which is contained in Z_j. Else, by choosing y' above y with the same distance between p and y than between p' and y', we will almost always get a node y' such that $y\ Z_j\ y'$. The only case where this won't work is if $p = c_{-1,k}$, $p' = c_{0,k}$ for some k. In that case, if y is at distance d of p, we will choose y' at distance $d + 2n + 3$ of p' instead. This is because the distance between bisimilar nodes around the starting points is smaller than elsewhere in the tree, so this bigger jump is required to close the gap.
  (3) Similarly, we can prove that for every $\star \in \{\downarrow^{-1}, \downarrow, (\downarrow^{-1})^+, \downarrow^+\}$, for every $p' \star y'$, there exists a $p \star y$ such that $y\ Z_j\ y'$. □

This contradicts the fact that φ was equivalent to the query (8.6), since $c_{-1,0}\ Z_n\ c_{0,0}$ but only $c_{0,0}$ satisfies the query (8.6). Hence, this query is not expressible in VerticalCoreXPath, and so it is not expressible in VerticalXPath.

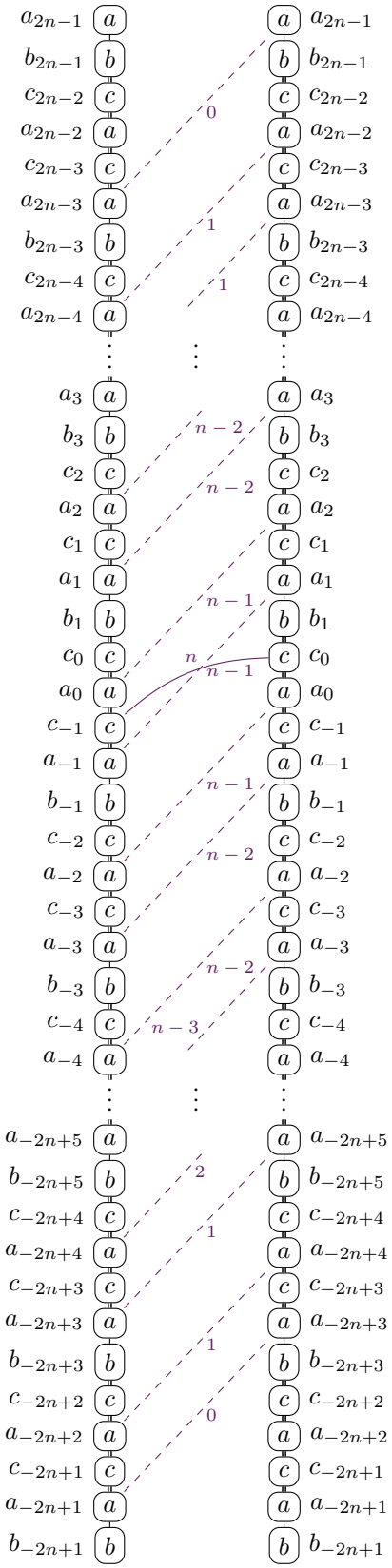

FIGURE A.1. n-bisimulation inside ℓ_n (a double edge represents a chain of n c-labelled nodes)

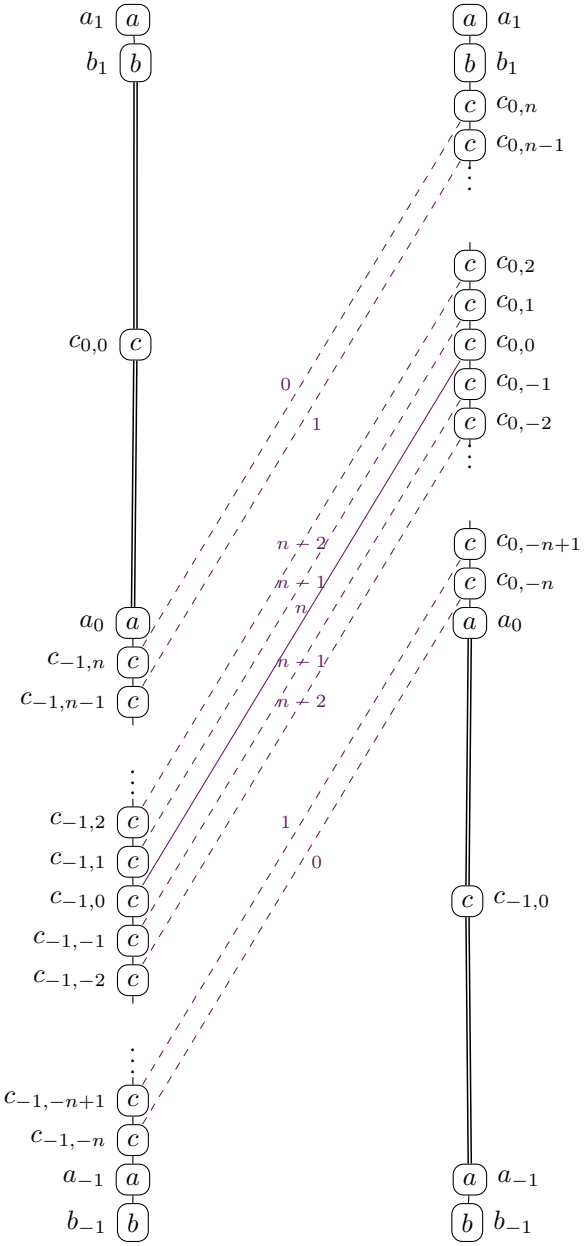

FIGURE A.2. Zoom around the starting points

A.1.2. Expressing last().

A.1.2.1. *Expressing* last() *in RegularXPath.* In this section, we show how to express $\texttt{last}(\pi)$ for *one-step* paths of the form $\pi = \alpha{::}*[\varphi]$. In some cases, it is easily expressible:

$$\begin{aligned}
\texttt{last}(\texttt{parent}{::}*[\varphi]) &\equiv \texttt{parent}{::}*[\varphi]\\
\texttt{last}(\texttt{self}{::}*[\varphi]) &\equiv \texttt{self}{::}*[\varphi]\\
\texttt{last}(\texttt{child}{::}*[\varphi]) &\equiv \texttt{child}{::}*[\varphi \text{ and } \texttt{not}(\texttt{following-sibling}{::}*[\varphi])]\\
\texttt{last}(\texttt{following-sibling}{::}*[\varphi]) &\equiv \texttt{following-sibling}{::}*[\varphi \text{ and }\\
&\qquad \texttt{not}(\texttt{following-sibling}{::}*[\varphi])]\\
\texttt{last}(\texttt{following}{::}*[\varphi]) &\equiv \texttt{following}{::}*[\varphi \text{ and } \texttt{not}(\texttt{following}{::}*[\varphi])\\
&\qquad \text{and } \texttt{not}(\texttt{descendant}{::}*[\varphi])]
\end{aligned}$$

The other cases can be handled by using RegularXPath:

$$\begin{aligned}
\texttt{last}(\texttt{ancestor}{::}*[\varphi]) &\equiv (\texttt{parent}{::}*[\texttt{not}(\varphi)])^*/\texttt{parent}{::}*[\varphi]\\
\texttt{last}(\texttt{descendant}{::}*[\varphi]) &\equiv (\texttt{child}{::}*[\texttt{descendant-or-self}{::}*[\varphi] \text{ and }\\
&\qquad \texttt{not}(\texttt{following-sibling}{::}*/\\
&\quad \texttt{descendant-or-self}{::}*[\varphi])])^+[\varphi \text{ and }\\
&\quad \texttt{not}(\texttt{descendant}{::}*[\varphi])]\\
\texttt{last}(\texttt{preceding-sibling}{::}*[\varphi]) &\equiv (\texttt{previous-sibling}{::}*[\texttt{not}(\varphi)])^*/\\
&\quad \texttt{previous-sibling}{::}*[\varphi]\\
\texttt{last}(\texttt{preceding}{::}*[\varphi]) &\equiv (\texttt{parent}{::}*[\texttt{not}((\texttt{previous-sibling}{::}*)^+/\\
&\quad (\texttt{child}{::}*)^*[\varphi])])^*/\\
&\quad (\texttt{previous-sibling}{::}*[\texttt{not}((\texttt{child}{::}*)^*[\varphi])])^+/\\
&\quad (\texttt{child}{::}*[\texttt{descendant-or-self}{::}*[\varphi] \text{ and }\\
&\qquad \texttt{not}(\texttt{following-sibling}{::}*/\\
&\quad \texttt{descendant-or-self}{::}*[\varphi])])^*[\varphi \text{ and }\\
&\quad \texttt{not}(\texttt{descendant}{::}*[\varphi])]
\end{aligned}$$

Note that the axis previous-sibling used above is not a proper XPath axis, but exists in RegularXPath. It corresponds to the relation $\rightarrow^{-1}$ of our models.

A.1.2.2. *Expressing notlast*() *in Core XPath.* In this section, we show how to express queries of the form *notlast*$(\alpha{::}*[\varphi])$ in CoreXPath 1.0.

$$\begin{aligned}
\textit{notlast}(\texttt{parent}{::}*[\varphi]) &\equiv \texttt{false}()\\
\textit{notlast}(\texttt{self}{::}*[\varphi]) &\equiv \texttt{false}()\\
\textit{notlast}(\texttt{child}{::}*[\varphi]) &\equiv \texttt{child}{::}*[\varphi \text{ and } \texttt{following-sibling}{::}*[\varphi]]\\
\textit{notlast}(\texttt{following-sibling}{::}*[\varphi]) &\equiv \texttt{following-sibling}{::}*[\varphi\\
&\qquad \text{and } \texttt{following-sibling}{::}*[\varphi]]
\end{aligned}$$

$$
\begin{aligned}
\mathit{notlast}(\texttt{following::}*[\varphi]) \equiv\ & \texttt{following::}*[\varphi \text{ and} \\
& (\texttt{following::}*[\varphi] \text{ or } \texttt{descendant::}*[\varphi])] \\
\mathit{notlast}(\texttt{ancestor::}*[\varphi]) \equiv\ & \texttt{ancestor::}*[\varphi]/\texttt{ancestor::}*[\varphi] \\
\mathit{notlast}(\texttt{preceding-sibling::}*[\varphi]) \equiv\ & \texttt{preceding-sibling::}*[\varphi]/ \\
& \texttt{preceding-sibling::}*[\varphi] \\
\mathit{notlast}(\texttt{descendant::}*[\varphi]) \equiv\ & \texttt{descendant::}*[\varphi \text{ and } \texttt{descendant::}*[\varphi]] \text{ union} \\
& \texttt{descendant::}*[\texttt{following-sibling::}*/ \\
& \qquad\qquad \texttt{descendant-or-self::}*[\varphi] \\
&]/\texttt{descendant-or-self::}*[\varphi] \\
\mathit{notlast}(\texttt{preceding::}*[\varphi]) \equiv\ & \texttt{preceding::}*[\varphi]/\texttt{preceding::}*[\varphi] \text{ union} \\
& \texttt{preceding::}*[\varphi \text{ and } \texttt{descendant::}*[\varphi]]
\end{aligned}
$$

A.1.3. Expressing `position()` **in Regular XPath.** In this section, we show how to translate predicates of the form α::*[φ][position() = i] in RegularXPath. This query selects the i-th node for the document order among the nodes that would have been selected by α::*[φ]. We represent such a predicate by a function pos_i. We give a translation for pos_i by induction on i.

$$
\begin{aligned}
\mathit{pos}_1(\texttt{parent::}*[\varphi]) \equiv\ & \texttt{parent::}*[\varphi] \\
\mathit{pos}_1(\texttt{self::}*[\varphi]) \equiv\ & \texttt{self::}*[\varphi] \\
\mathit{pos}_1(\texttt{child::}*[\varphi]) \equiv\ & \texttt{child::}*[\varphi \text{ and } \texttt{not}(\texttt{preceding-sibling::}*[\varphi])] \\
\mathit{pos}_1(\texttt{preceding-sibling::}*[\varphi]) \equiv\ & \texttt{preceding-sibling::}*[\varphi \text{ and} \\
& \texttt{not}(\texttt{preceding-sibling::}*[\varphi])] \\
\mathit{pos}_1(\texttt{following-sibling::}*[\varphi]) \equiv\ & (\texttt{next-sibling::}*[\texttt{not}(\varphi)])^*/\texttt{next-sibling::}*[\varphi] \\
\mathit{pos}_1(\texttt{ancestor::}*[\varphi]) \equiv\ & \texttt{ancestor::}*[\varphi \text{ and } \texttt{not}(\texttt{ancestor::}*[\varphi])] \\
\mathit{pos}_1(\texttt{descendant::}*[\varphi]) \equiv\ & (\texttt{child::}*[\varphi \text{ and } \texttt{descendant::}*[\varphi] \text{ and} \\
& \quad \texttt{not}(\texttt{preceding-sibling::}*/ \\
& \qquad \texttt{descendant-or-self::}*[\varphi]) \\
&])^*/\texttt{child::}*[\varphi \text{ and } \texttt{not}(\texttt{preceding-sibling::}*/ \\
& \texttt{descendant-or-self::}*[\varphi])] \\
\mathit{pos}_1(\texttt{following::}*[\varphi]) \equiv\ & \texttt{ancestor-or-self::}*[\texttt{following::}*[\varphi] \text{ and} \\
& \texttt{not}(\texttt{parent::}*/\texttt{following::}*[\varphi])]/ \\
& \mathit{pos}_1(\texttt{following-sibling::}* \\
& \quad [\texttt{descendant-or-self::}*[\varphi]])/ \\
& (.[\varphi] \text{ union } .[\texttt{not}(\varphi)]/\mathit{pos}_1(\texttt{descendant::}*[\varphi]))
\end{aligned}
$$

$$\begin{aligned}
pos_1(\texttt{preceding::}*[\varphi]) \equiv\ & \texttt{ancestor-or-self::}*[\texttt{preceding::}*[\varphi]\ \texttt{and}\\
& \texttt{not(parent::}*\texttt{/preceding::}*[\varphi]\texttt{)]/}\\
& pos_1(\texttt{preceding-sibling::}*\\
& \quad [\texttt{descendant-or-self::}*[\varphi]])/\\
& (.[\varphi]\ \texttt{union}\ .[\texttt{not}(\varphi)]/pos_1(\texttt{descendant::}*[\varphi]))
\end{aligned}$$

$$\begin{aligned}
pos_{i+1}(\texttt{parent::}*[\varphi]) &\equiv \texttt{false()}\\
pos_{i+1}(\texttt{self::}*[\varphi]) &\equiv \texttt{false()}\\
pos_{i+1}(\texttt{child::}*[\varphi]) &\equiv pos_i(\texttt{child::}*[\varphi])/pos_1(\texttt{following-sibling::}*[\varphi])\\
pos_{i+1}(\texttt{following-sibling::}*[\varphi]) &\equiv pos_i(\texttt{following-sibling::}*[\varphi])/\\
&\quad pos_1(\texttt{following-sibling::}*[\varphi])\\
pos_{i+1}(\texttt{preceding-sibling::}*[\varphi]) &\equiv pos_1(\texttt{preceding-sibling::}*[\varphi\ \texttt{and}\\
&\quad pos_i(\texttt{preceding-sibling::}*[\varphi])])\\
pos_{i+1}(\texttt{ancestor::}*[\varphi]) &\equiv pos_1(\texttt{ancestor::}*[\varphi\ \texttt{and}\\
&\quad pos_i(\texttt{ancestor::}*[\varphi])])\\
pos_{i+1}(\texttt{descendant::}*[\varphi]) &\equiv pos_i(\texttt{descendant::}*[\varphi])[\texttt{descendant::}*[\varphi]]/\\
&\quad pos_1(\texttt{descendant::}*[\varphi])\ \texttt{union}\\
&\quad pos_i(\texttt{descendant::}*[\varphi])[\texttt{not(descendant::}*[\varphi]\texttt{)]}/\\
&\quad pos_1(\texttt{following::}*[\varphi])\\
pos_{i+1}(\texttt{following::}*[\varphi]) &\equiv pos_i(\texttt{following::}*[\varphi])[\texttt{descendant::}*[\varphi]]/\\
&\quad pos_1(\texttt{descendant::}*[\varphi])\ \texttt{union}\\
&\quad pos_i(\texttt{following::}*[\varphi])[\texttt{not(descendant::}*[\varphi]\texttt{)]}/\\
&\quad pos_1(\texttt{following::}*[\varphi])\\
pos_{i+1}(\texttt{preceding::}*[\varphi]) &\equiv \Pi_i\ \texttt{union}\ .[\texttt{not}(\Pi_i)]/pos_1(\texttt{preceding::}*[\varphi\ \texttt{and}\\
&\quad pos_i(\texttt{preceding::}*[\varphi])])
\end{aligned}$$

With :

$$\Pi_i = \underset{1\leq j\leq i}{\texttt{union}}\ pos_j(\texttt{preceding::}*[\varphi])/pos_{i+1-j}(\texttt{descendant::}*[\varphi])$$

A.2. Decidability Results on id()

A.2.1. Reducing from PCP Using Data Joins. In this section, we show that the satisfiability problem is undecidable for DownwardXPath+id. This will entail undecidability for ForwardXPath+id and VerticalXPath+id as well.

Let Σ be an alphabet, and let $\{(u_i, v_i) \mid 0 \leq i \leq n\} \subseteq (\Sigma^+)^2$ be a PCP instance over Σ. We note $\overline{\Sigma} = \{\overline{a} \mid a \in \Sigma\}$, and we give ourselves a fresh symbol # that will mark the start of a PCP domino. We will encode a potential solution to this PCP instance as a branch in a data tree over $\Sigma \cup \overline{\Sigma}$ for which the corresponding word will belong to $\{\#u_i\overline{v_i} \mid 0 \leq i \leq n\}^+$. Remark

that, unlike a perhaps common mistaken belief, the XML specification does not force the @id attributes of a document to be unique. The encoding we present will however force them to be unique along a branch. Nonetheless, since we are working with the DownwardXPath fragment, we cannot prevent the models from having other branches, but we will be able to force all the branches to be identical. Note that our encoding could be simplified if we assumed that the @id attributes are unique, as some part of our formulæ would then be trivially satisfied.

We will use the id links to move around in this encoding: let $\tilde{w}$ be the word corresponding to our encoding branch of the tree, and let w (resp. $\overline{w}$) be the longest subword of $\tilde{w}$ in Σ (resp. $\overline{\Sigma}$). Every node w_i will have an attribute @succ for which the datum will match the @id key of the node w_{i+1}, and an attribute @sym for which the datum will match the @id key of the node $\overline{w}_i$. The nodes $\overline{w}_i$ will also have attributes @succ and @sym playing similar roles. Such a word $\tilde{w}$ will be the encoding of a solution of our PCP instance if and only if $w = \overline{w}$.

Hence, we need to find formulas that will force this encoding to be respected. At the end, a data tree will satisfy this set of formulas if and only if it encodes a solution to our PCP instance.

First, we can force that every node has some attributes @id, @succ and @sym:

$$\Psi_0 := \texttt{not}(//*[\texttt{not}(@\texttt{id}) \texttt{ or not}(@\texttt{succ}) \texttt{ or not}(@\texttt{sym})]$$

We now force all the branches from the root to be identical:

- the @id keys are unique along a branch:

$$\Psi_1 := \texttt{not}(//*[@\texttt{id eq descendant::}*/@\texttt{id}])$$

- if two nodes share the same @id key, then all their children are labelled by the same letter:

$$\Psi_2 := \bigwedge_{a \neq b \in \Sigma \cup \overline{\Sigma} \cup \{\#\}} \texttt{not}(//*[\texttt{id}(@\texttt{id})/\texttt{child::}a \texttt{ and id}(@\texttt{id})/\texttt{child::}b])$$

- if two nodes share the same @id key, then all their children have the same @id key:

$$\Psi_3 := \texttt{not}(//*[\texttt{id}(@\texttt{id})/\texttt{child::}*/@\texttt{id ne id}(@\texttt{id})/\texttt{child::}*/@\texttt{id}])$$

- if two nodes share the same @id key, then they have the same @sym data value and the same @succ data value:

$$\Psi_4 := \texttt{not}(//*[(\texttt{id}(@\texttt{id})/@\texttt{sym ne id}(@\texttt{id})/@\texttt{sym}) \texttt{ or } (\texttt{id}(@\texttt{id})/@\texttt{succ ne id}(@\texttt{id})/@\texttt{succ})])$$

- a node has a child iff all the nodes sharing its @id key have a child:

$$\Psi_5 := \texttt{not}(//*[\texttt{id}(@\texttt{id})[\texttt{child::}*] \texttt{ and id}(@\texttt{id})[\texttt{not}(\texttt{child::}*)]])$$

Now, we can prove the following lemma:

LEMMA A.2. *Let t be a data tree. Then, t satisfies all the Ψ_i formulas if and only if:*

(1) All the nodes of t at the same depth level are identical (same label, same @id, @succ *and* @sym *values) and have children with the same label and the same* @id *key.*

(2) Two nodes from different depth levels do not share the same @id *key.*

PROOF. It is easy to see that a data tree satisfying the properties 1 and 2 will satisfy the formulas Ψ_i. Suppose now that a data tree t satisfies the formulas Ψ_i. We prove that the properties 1 and 2 are satisfied by induction on the depth level:

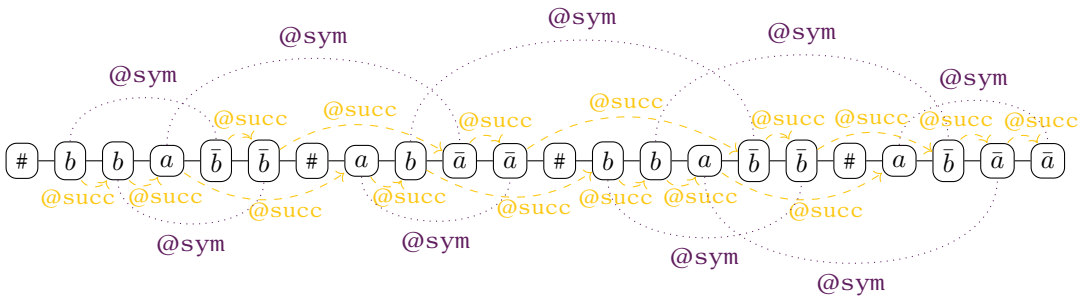

FIGURE A.3. Example of the encoding of a PCP solution.

- Level depth 0: there is only the root node at this depth level. Thanks to the formula Ψ_1, and since the root node appears in every branch of the tree, there is no node somewhere else in the tree with the same @id key than the root so the property 2 is satisfied. And now, thanks to Ψ_2, all its children have the same label, and thanks to Ψ_3, all its children have the same @id key. So the property 1 is also satisfied.
- Let us assume that the properties 1 and 2 are satisfied down to depth level n, and let us prove they are also satisfied at depth level $n+1$. Since the nodes of depth n satisfy the property 1, then all the nodes of depth $n+1$ have the same label and the same @id key (they are children from nodes of depth n). Then, thanks to Ψ_4, they also have the same @succ and @sym values, so they are identical. Moreover, thanks to Ψ_2 and Ψ_3, all their children have the same label and @id value (because of the nodes of depth $n+1$ have the same @id key). So the property 1 is satisfied at depth level $n+1$. Furthermore, thanks to Ψ_1, no node above or below a node of depth $n+1$ (that is, no node from a different depth level) share the same @id key than the nodes of depth $n+1$, so the property 2 is also satisfied.

Thus, by induction on the depth level, the properties 1 and 2 are satisfied everywhere in the tree. □

This property allows us to only consider non-branching models, as any model will be logically equivalent to one of its branches for DownwardXPath+id.

EXAMPLE A.3. Let us consider the following PCP instance, where $\Sigma = \{a, b\}$. $(u_1, \overline{v_1}) = (a, \overline{baa})$, $(u_2, \overline{v_2}) = (ab, \overline{aa})$, $(u_3, \overline{v_3}) = (bba, \overline{bb})$. The sequence $(3, 2, 3, 1)$ is a solution, and its encoding is represented in Figure A.3.

We start by giving ourselves formulas expressing simple properties.

- This formula checks that the label of the current node is in $\Sigma = \{a_1, \dots a_n\}$:

$$\varphi_\Sigma := \texttt{self::}a_1 \texttt{ union } \dots \texttt{ union self::}a_n$$

- Symmetrically, we define the formula $\varphi_{\overline{\Sigma}}$.
- This formula checks that the current node is a marker at the beginning of a group $u_i\overline{v_i}$:

$$\varphi_{first} := \texttt{self::\#}$$

- Finally, this formula checks that the current node is the last letter of a group $u_i\overline{v_i}$:

$$\varphi_{last} := \texttt{not(child::}*\texttt{) or child::\#}$$

We can now express the marker property of the # symbol:

$$\Phi_1 := \texttt{not}(//*[\varphi_\Sigma]/\texttt{child::\#}) \text{ and } \texttt{not}(//\#/\texttt{child::}*[\varphi_{\overline{\Sigma}}]) \text{ and } \texttt{not}(//\#/\#)$$

Moreover, we can express that a given word appears correctly between markers, somewhere in the tree. Let $u_i\overline{v_i} = c_{i,0}\dots c_{i,m_i}$. We define the formula φ_i that checks that the current node is a marker #, followed by letters forming the word $u_i\overline{v_i}$, and that this word is not followed by extra letters:

$$\varphi_i := \varphi_{first} \text{ and } ./c_{i,0}/c_{i,1}/\dots/(c_{i,m_i}[\varphi_{last}])$$

Thus, we can express the fact that the word w corresponding to any branch of a model is in $\{\#u_i\overline{v_i} \mid 0 \leq i \leq n\}^+$:

$$\Phi_2 := /*[\varphi_{first}] \text{ and } \texttt{not}(//*[\varphi_{first} \text{ and } \texttt{not}(\varphi_0) \text{ and } \texttt{not} \dots \text{ and } \texttt{not}(\varphi_n)])$$

Then, we must express the encoding properties about the `id` links:

- every letter has a successor except the last letter of the longest subwords from Σ and $\overline{\Sigma}$:

$$\begin{aligned}\Phi_3 := \texttt{not}(//*[\texttt{not}(\texttt{@succ}) \text{ and } ((\varphi_\Sigma \text{ and } \texttt{descendant::}*[\varphi_\Sigma]) \text{ or}\\ (\varphi_{\overline{\Sigma}} \text{ and } \texttt{descendant::}*[\varphi_{\overline{\Sigma}}]))]) \text{ and}\\ \texttt{not}(//*[\texttt{@succ and } ((\varphi_\Sigma \text{ and } \texttt{not}(\texttt{descendant::}*[\varphi_\Sigma])) \text{ or}\\ (\varphi_{\overline{\Sigma}} \text{ and } \texttt{not}(\texttt{descendant::}*[\varphi_{\overline{\Sigma}}])))])\end{aligned}$$

- every letter has a `@sym` attribute:

$$\Phi_4 := \texttt{not}(//*[\texttt{not}(\texttt{@sym}) \text{ and } \texttt{not}(\texttt{self::\#})])$$

- `id(@succ)` is non decreasing:

$$\Phi_5 := \texttt{not}(//*[\texttt{@succ and not}(\texttt{@succ eq descendant::}*/\texttt{@id})])$$

- `id(@succ)` doesn't link letters from Σ with letters from $\overline{\Sigma}$:

$$\Phi_6 := \texttt{not}(//*[(\varphi_\Sigma \text{ and } \texttt{id(@succ)}[\varphi_{\overline{\Sigma}}]) \text{ or } (\varphi_{\overline{\Sigma}} \text{ and } \texttt{id(@succ)}[\varphi_\Sigma])])$$

- `id(@succ)` doesn't jump over a letter from the same alphabet:

$$\begin{aligned}\Phi_7 := \texttt{not}(//*[\varphi_\Sigma \text{ and } \texttt{@succ eq descendant::}*[\varphi_\Sigma]/\texttt{descendant::}*/\texttt{@id}]) \text{ and}\\ \texttt{not}(//*[\varphi_{\overline{\Sigma}} \text{ and } \texttt{@succ eq descendant::}*[\varphi_{\overline{\Sigma}}]/\texttt{descendant::}*/\texttt{@id}])\end{aligned}$$

- `id(@sym)` links properly a letter to its bar version:

$$\begin{aligned}\Phi_8 := \bigwedge_{a\in\Sigma}(\texttt{not}(//*[\texttt{self::}a \text{ and } \texttt{not}(\texttt{id(@sym)/self::}\overline{a})]) \text{ and}\\ \texttt{not}(//*[\texttt{self::}\overline{a} \text{ and } \texttt{not}(\texttt{id(@sym)/self::}a)]))\end{aligned}$$

- `id(@sym)` is an involution:

$$\Phi_9 := \texttt{not}(//*[\texttt{not}(\texttt{id(sym)/@sym eq @id})])$$

- compatibility of `id(@succ)` and `id(@sym)` (cf. Figure A.4):

$$\Phi_{10} := \texttt{not}(//*[\texttt{@succ and not}((\texttt{id(@succ)/@sym eq id(@sym)/@succ}))])$$

We are now ready to prove our reduction to PCP.

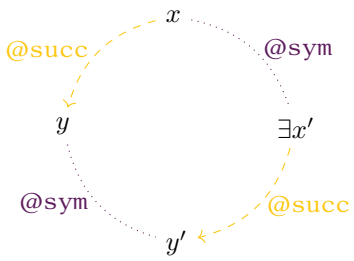

FIGURE A.4. Φ_{10} expresses the compatibility between $\texttt{id}(\texttt{@succ})$ and $\texttt{id}(\texttt{@sym})$: if x has a successor y, then the symmetric of x must also have a successor, which is the symmetric of y.

PROPOSITION A.4. *An instance of PCP has a solution if and only if its encoding in Downward*XPath+$\texttt{id}$ *is satisfiable.*

PROOF. It is easy to check that if the PCP instance has a solution, encoding it as described above will give us a data tree satisfying all the formulas. Conversely, let us prove that if all these formulas are satisfied by a data tree t, then we can extract a solution of our PCP instance from t.

First, thanks to lemma A.2, we can assume for the sake of simplicity that t is a non-branching tree. Then, because t satisfies Φ_1 and Φ_2, the word $\tilde{w}$ of t belongs to $\{\#u_i\overline{v_i}\}^+$. Let us call w (resp. $\overline{w}$) the longest subword of t in Σ (resp. $\overline{\Sigma}$). Now, thanks to Φ_3 every letter of w and $\overline{w}$ has a @succ attribute (except their last letter), and thanks to Φ_4 every letter of w and $\overline{w}$ has a @sym attribute. Moreover, because t satisfies Φ_5, Φ_6 and Φ_7, we can prove that the @succ links are jumping as intended from a letter of w (resp. $\overline{w}$) to the next one. This give us two @succ chains which correspond to w and $\overline{w}$. We note $x\ \mathcal{R}_{succ}\ y$ if there is a @succ link from x to y.

We now need to check the @sym links. These links are forming a symmetric relation (thanks to Φ_9) that we will denote by $\mathcal{R}_{sym}$. First, let us consider the beginning of w: let j be the position in $\overline{w}$ such that $w_0\ \mathcal{R}_{sym}\ \overline{w}_j$, and let us assume that $j > 0$. Then, there is a node $\overline{w}_{j-1}$ above in t such that $\overline{w}_{j-1}\ \mathcal{R}_{succ}\ \overline{w}_j$. Now, because of the compatibility property expressed by Φ_{10}, there should be a node x above w_0 in the tree such that $\overline{w}_{j-1}\ \mathcal{R}_{sym}\ x\ \mathcal{R}_{succ}\ w_0$, which is not possible by definition of w_0. So we have $w_0\ \mathcal{R}_{sym}\ \overline{w}_0$. Now, thanks to the same compatibility property, we can prove by induction on i that we have $w_i\ \mathcal{R}_{sym}\ \overline{w}_i$ for every i.

What could still happen is that w and $\overline{w}$ are not of the same length. This is also forbidden by Φ_{10}: let w_{n-1} be the last letter of w, and let us assume that $\overline{w}$ is of length $\geq n$ (the other case is symmetric). Then we have $w_{n-1}\ \mathcal{R}_{sym}\ \overline{w}_{n-1}$, and now if $\overline{w}_{n-1}$ had a successor $\overline{w}_n$, then by using Φ_{10} we could prove that there should be a node x such that $w_{n-1}\ \mathcal{R}_{succ}\ x\ \mathcal{R}_{sym}\ \overline{w}_n$, which is not possible by assumption that w_{n-1} was the last letter of w. So w and $\overline{w}$ are of the same length.

Finally, thanks to Φ_8 we can prove that for $0 \leq i < n, \overline{(w_i)} = \overline{w}_i$, so t does represent a solution to our PCP instance. □

This entails the undecidability of the satisfiability problem for the following fragments:

COROLLARY A.5. *The satisfiability problem is undecidable for DownwardXPath+*id*, ForwardXPath+*id *and VerticalXPath+*id.

A.2.2. Reducing from PCP using Node Tests. The encoding above can be easily adapted for a restricted version of CoreXPath 2.0 with only the downward axes, by replacing the data joins by node tests using is. Here is how one can transform the previous formulas to be in such a fragment:

(1) Since every data value is actually the @id key of a node, every data test of the form $\pi/@\texttt{foo eq } \pi'$ can be transformed into $\pi/\texttt{id}(@\texttt{foo})/@\texttt{id eq } \pi'$.
(2) Once all the data joins are of the form $\pi/@\texttt{id eq } \pi'/@\texttt{id}$, we can replace them by node tests of the form $\pi \texttt{ is } \pi'$ (since all the @id keys are unique).

Hence, our undecidability result also holds for CoreXPath 2.0.

COROLLARY A.6. *The satisfiability problem is undecidable for CoreXPath 2.0+*id.

A.2.3. Decidable Fragments with id**.** In this section, we show how we can encode some uses of id by using variables axiomatised with data joins. Let us recall what uses of id we allowed in VerticalXPath extended with variables, and EMSO^{2}XPath:

$$\pi ::= \cdots \mid /\pi_{id}$$
$$\pi_{id} ::= \pi \mid \pi_{id}/\pi_{id} \mid \pi_{id} \texttt{ union } \pi_{id} \mid \pi_{id}[\varphi] \mid \texttt{id}(\pi_{id})$$

When an occurrence of id() is applied at the end of an absolute path $/\Pi$, it is translated into a variable $\$x_\Pi$ that will be forced to contain exactly the nodes to which the id() call would jump, using the following axioms:

- $\varphi_1(\$x_\Pi, \Pi) := \texttt{not}(/\Pi[(\texttt{not}(. \texttt{ eq } \$x_\Pi/@\texttt{id})) \texttt{ and } . \texttt{ eq } //*/@\texttt{id}])$
- $\varphi_2(\$x_\Pi, \Pi) := \texttt{not}(\$x_\Pi[\texttt{not}(@\texttt{id eq } /\Pi)])$

We will eliminate the id occurrences by using a translation function $\mathsf{T}_{\texttt{id}}$ taking two arguments: the second one is the query that remains to be translated, and the first one is a leftover path π, not containing any id() call, that will be used to axiomatise variables $\$x_\pi$. In the following translation, whenever a formula denoted $\$x_\Pi$ is used, the corresponding axioms $\varphi_1(\$x_\Pi, \Pi)$ and $\varphi_2(\$x_\Pi, \Pi)$ will be added at top level.

$$\begin{aligned}
\mathsf{T}_{\texttt{id}}(\pi, \pi'/\pi_{id}) &= \mathsf{T}_{\texttt{id}}(\pi/\pi', \pi_{id}) && \text{(where } \pi\text{' does not contain any id() call)}\\
\mathsf{T}_{\texttt{id}}(\pi, \pi_{id}/\pi'_{id}) &= \mathsf{T}_{\texttt{id}}(\mathsf{T}_{\texttt{id}}(\pi, \pi_{id}), \pi'_{id})\\
\mathsf{T}_{\texttt{id}}(\pi, \texttt{id}(\pi')) &= \$x_{\pi/\pi'}\\
\mathsf{T}_{\texttt{id}}(\pi, \texttt{id}(\pi_{id})) &= \mathsf{T}_{\texttt{id}}(\varepsilon, \texttt{id}(\mathsf{T}_{\texttt{id}}(\pi, \pi_{id})))\\
\mathsf{T}_{\texttt{id}}(\pi, \pi') &= \pi/\pi' && \text{(where } \pi\text{' does not contain any id() call)}\\
\mathsf{T}_{\texttt{id}}(\pi, \pi_{id} \texttt{ union } \pi'_{id}) &= \mathsf{T}_{\texttt{id}}(\pi, \pi_{id}) \texttt{ union } \mathsf{T}_{\texttt{id}}(\pi, \pi'_{id})\\
\mathsf{T}_{\texttt{id}}(\pi, \pi'[\varphi]) &= \mathsf{T}_{\texttt{id}}(\pi, \pi')[\mathsf{T}_{\texttt{id}}(\varepsilon, \varphi)]\\
\mathsf{T}_{\texttt{id}}(\varepsilon, \varphi \texttt{ and } \varphi') &= \mathsf{T}_{\texttt{id}}(\varepsilon, \varphi) \texttt{ and } \mathsf{T}_{\texttt{id}}(\varepsilon, \varphi')\\
\mathsf{T}_{\texttt{id}}(\varepsilon, \varphi \texttt{ or } \varphi') &= \mathsf{T}_{\texttt{id}}(\varepsilon, \varphi) \texttt{ or } \mathsf{T}_{\texttt{id}}(\varepsilon, \varphi')\\
\mathsf{T}_{\texttt{id}}(\varepsilon, \texttt{not}(\varphi)) &= \texttt{not}(\mathsf{T}_{\texttt{id}}(\varepsilon, \varphi))
\end{aligned}$$

We denote by Ax all the axioms that must be added at top level:

$$\mathrm{Ax}(\mathsf{T}_{\mathtt{id}}(\pi,\tilde{\pi})) = \bigwedge_{\$\mathrm{x}_\Pi \text{ appearing in } \mathsf{T}_{\mathtt{id}}(\pi,\tilde{\pi})} \varphi_1(\$\mathrm{x}_\Pi,\Pi) \wedge \varphi_2(\$\mathrm{x}_\Pi,\Pi)$$

And we note $t,\epsilon,n_f \vDash_{Ax} \mathsf{T}_{\mathtt{id}}(\pi,\tilde{\pi})$ when $t,\epsilon,n_f \vDash \mathsf{T}_{\mathtt{id}}(\pi,\tilde{\pi}) \wedge \mathrm{Ax}(\mathsf{T}_{\mathtt{id}}(\pi,\tilde{\pi}))$

Proposition A.7. *Let t be a data tree and let $n_f \in N$. Let π and $\tilde{\pi}$ be path formulas in our fragment extended with* id*. We have:*

$$t,\epsilon,n_f \vDash_{Ax} \mathsf{T}_{\mathtt{id}}(\pi,\tilde{\pi}) \textit{ iff } t,\epsilon,n_f \vDash \pi/\tilde{\pi}$$

Proof. We prove this theorem by induction on the lexicographic order over the number of occurrences of $\mathtt{id}()$ in $\tilde{\pi}$ and the size of $\tilde{\pi}$, showing only the non trivial cases:

- case $\tilde{\pi} = \pi_{id}/\pi'_{id}$:

$$\begin{aligned}
& t,\epsilon,n_f \vDash_{Ax} \mathsf{T}_{\mathtt{id}}(\pi,\pi_{id}/\pi'_{id}) \\
\iff & t,\epsilon,n_f \vDash_{Ax} \mathsf{T}_{\mathtt{id}}(\mathsf{T}_{\mathtt{id}}(\pi,\pi_{id}),\pi'_{id}) \\
\iff & t,\epsilon,n_f \vDash_{Ax} \mathsf{T}_{\mathtt{id}}(\pi,\pi_{id})/\pi'_{id} \\
\iff & \exists n \in N\ t,\epsilon,n \vDash_{Ax} \mathsf{T}_{\mathtt{id}}(\pi,\pi_{id}) \text{ and } t,n,n_f \vDash \pi'_{id} \\
\iff & \exists n \in N\ t,\epsilon,n \vDash \pi/\pi_{id} \text{ and } t,n,n_f \vDash \pi'_{id} \\
\iff & t,\epsilon,n_f \vDash \pi/\pi_{id}/\pi'_{id}
\end{aligned}$$

- case $\tilde{\pi} = \mathtt{id}(\pi_{id})$:

$$\begin{aligned}
& t,\epsilon,n_f \vDash_{Ax} \mathsf{T}_{\mathtt{id}}(\pi,\mathtt{id}(\pi_{id})) \\
\iff & t,\epsilon,n_f \vDash_{Ax} \mathsf{T}_{\mathtt{id}}(\varepsilon,\mathtt{id}(\mathsf{T}_{\mathtt{id}}(\pi,\pi_{id}))) \\
\iff & t,\epsilon,n_f \vDash_{Ax} \mathtt{id}(\mathsf{T}_{\mathtt{id}}(\pi,\pi_{id})) \\
\iff & \exists n_{id} \in N,\ n_f \downarrow n_{id} \text{ and } \exists n'_f \in N,\ \delta(n_{id}) = \delta(n'_f) \text{ and } t,\epsilon,n'_f \vDash_{Ax} \mathsf{T}_{\mathtt{id}}(\pi,\pi_{id}) \\
\iff & \exists n_{id} \in N,\ n_f \downarrow n_{id} \text{ and } \exists n'_f \in N,\ \delta(n_{id}) = \delta(n'_f) \text{ and } t,\epsilon,n'_f \vDash \pi/\pi_{id} \\
\iff & \exists n_{id} \in N,\ n_f \downarrow n_{id} \text{ and } \exists n'_f \in N,\ \delta(n_{id}) = \delta(n'_f) \\
& \quad \text{and } \exists n \in N, t,\epsilon,n \vDash \pi \text{ and } t,n,n'_f \vDash \pi_{id} \\
\iff & \exists n \in N\ t,\epsilon,n \vDash \pi \text{ and } t,n,n_f \vDash \mathtt{id}(\pi_{id}) \\
\iff & t,\epsilon,n_f \vDash \pi/\mathtt{id}(\pi_{id})
\end{aligned}$$

- case $\tilde{\pi} = \mathtt{id}(\pi')$ (with no occurrence of $\mathtt{id}()$ in π'):

$$\begin{aligned}
& t,\epsilon,n_f \vDash_{Ax} \mathsf{T}_{\mathtt{id}}(\pi,\mathtt{id}(\pi')) \\
\iff & t,\epsilon,n_f \vDash_{Ax} \$\mathrm{x}
\end{aligned}$$

where $\$\mathrm{x}$ is a fresh variable axiomatised at top level with:
(1) $\varphi_1 := \mathtt{not}(/\pi/\pi'[(.\ \mathtt{eq}\ \$\mathrm{x}/@\mathtt{id})\ \mathtt{and}\ .\ \mathtt{eq}\ //*/@\mathtt{id}])$
(2) $\varphi_2 := \mathtt{not}(\$\mathrm{x}[\mathtt{not}(@\mathtt{id}\ \mathtt{eq}\ /\pi/\pi')])$
The formula φ_1 ensures that $[\![/\pi/\mathtt{id}(\pi')]\!]^{\nu}_{\mathrm{p}} \subseteq [\![\$\mathrm{x}]\!]^{\nu}_{\mathrm{p}}$, and φ_2 ensures that $[\![\$\mathrm{x}]\!]^{\nu}_{\mathrm{p}} \subseteq [\![/\pi/\mathtt{id}(\pi')]\!]^{\nu}_{\mathrm{p}}$. Hence, the equivalence is true. □

In addition, for the EMSO^{2}XPath fragment only, we can handle paths that do not start from the root as we can use a second-order variable to remember the starting point. In the translation of a query in an EMSO^2 formula, if the initial node is denoted by a first-order variable x, we can axiomatise a second-order variable X_{start} as follows:

$$x \in X_{start} \wedge \forall y, y \in X_{start} \Rightarrow y = x$$

Then, if an `id()` is used in a non-rooted path, we can handle it by adapting the previous axiomatisation and testing for X_{start} instead of testing for the root:

(1) $\texttt{not}(\$\texttt{x}_{start}/\Pi[(\texttt{not}(.\ \texttt{eq}\ \$\texttt{x}_{\Pi}/@\texttt{id}))\ \texttt{and}\ .\ \texttt{eq}\ //*/@\texttt{id}])$
(2) $\texttt{not}(\$\texttt{x}_{\Pi}[\texttt{not}(@\texttt{id}\ \texttt{eq}\ \$\texttt{x}_{start}/\Pi)])$

But we must be careful not to allow such occurrences of `id()` in a node expression, as the Example A.8 shows. In that sense, we must provide our grammar with a new starting symbol π' that can either be a path π as defined previously, or a non-rooted path π_{id} using `id()`:

$$\pi' ::= \pi \mid \pi_{id}$$

Example A.8. The formula $/\texttt{child}::a/\texttt{id}(@\texttt{x})$ will be translated to $\$\texttt{x}$ by our standard translation, or could also be translated to $/\texttt{child}::a/\$\texttt{x}$ if we want to simulate the path it follows more accurately, where the variable $\$\texttt{x}$ axiomatised by:

(1) $\texttt{not}(/\texttt{child}::a/@\texttt{x}[\texttt{not}(.\ \texttt{eq}\ \$\texttt{x}/@\texttt{id})\ \texttt{and}\ .\ \texttt{eq}\ //*/@\texttt{id}])$
(2) $\texttt{not}(\$\texttt{x}[\texttt{not}(@\texttt{id}\ \texttt{eq}\ /\texttt{child}::a/@\texttt{x})])$

However, even though the translation will select the same nodes as the original query, the variable $\$\texttt{x}$ will do a global jump directly to all the results, no matter what the current node is (cf. Figure A.5). Because of this, we cannot allow the use of `id` in a test: a test of the form $[\texttt{id}(@\texttt{x})]$ will be satisfied if a jump $\texttt{id}(@\texttt{x})$ can be done from the current node, but a test of the form $[\$\texttt{x}]$ will be satisfied if the variable $\$\texttt{x}$ is not empty, no matter what the current node is (cf. Figure A.6).

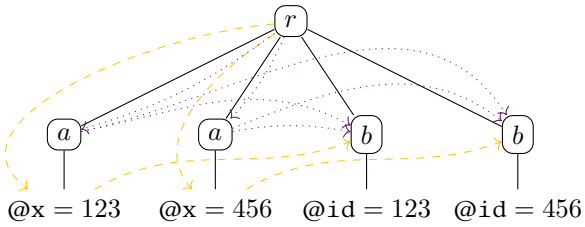

Figure A.5. The paths followed by the query $/\texttt{child}::a/\texttt{id}(@\texttt{x})$ in yellow, and by the query $/\texttt{child}::a/\$\texttt{x}$ in violet.

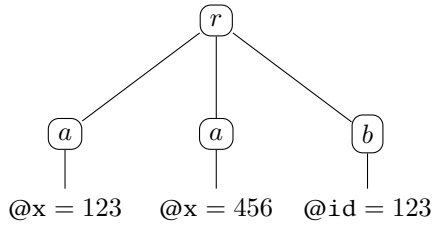

FIGURE A.6. Counter example of a path formula using a non-rooted `id` in a node test for which our translation would fail. The query $/\texttt{child}::a[\texttt{id}(@\texttt{x})]$ would only select the leftmost child of the root, but the query $/\texttt{child}::a[\$\texttt{x}]$ with $\$\texttt{x}$ axiomatised as above would select all the children labelled by a, since $\$\texttt{x}$ would be non empty.

Bibliography

XML path language (XPath) version 1.0. W3C Recommendation, 1999. URL https://www.w3.org/TR/1999/REC-xpath-19991116/. Cited on page 2.

RELAX NG compact syntax. OASIS Committee Specification, 2002. URL http://relaxng.org/compact.html. Cited on page 85.

Extensible Markup Language (XML) 1.0. W3c recommendation, 2008. URL https://www.w3.org/TR/xml/. Cited on page 1.

XPath and XQuery functions and operators 3.0. W3C Recommendation, 2014. URL https://www.w3.org/TR/xpath-functions-30/. Cited on pages 77, 80, 82, 85, 98, and 100.

XML path language (XPath) 3.0. W3C Recommendation, 2014. URL http://www.w3.org/TR/xpath-3/. Cited on pages 2 and 83.

XQuery and XPath data model 3.0. W3C Recommendation, 2014. URL http://www.w3.org/TR/xpath-datamodel-30/. Cited on page 83.

XQuery 3.0: An XML query language. W3C Recommendation, 2014a. URL http://www.w3.org/TR/xquery-30/. Cited on page 2.

XQueryX 3.0. W3C Recommendation, 2014b. URL http://www.w3.org/TR/xqueryx-3/. Cited on pages 78, 84, and 85.

XSL Transformations (XSLT) version 3.0. W3C Recommendation, 2017. URL http://www.w3.org/TR/xslt-30/. Cited on page 2.

P. A. Abdulla, M. F. Atig, Y.-F. Chen, L. Holík, A. Rezine, P. Rümmer, and J. Stenman. Norn: An SMT solver for string constraints. In *CAV '15*, volume 9206 of *Lect. Notes in Comput. Sci.*, pages 462–469. Springer, 2015. doi:10.1007/978-3-319-21690-4_29. Cited on page 113.

S. Abriola, M. E. Descotte, R. Fervari, and S. Figueira. Axiomatizations for downward XPath on data trees. *J. Comput. Syst. Sci.*, 89:209–245, 2017a. doi:10.1016/j.jcss.2017.05.008. Cited on page 7.

S. Abriola, D. Figueira, and S. Figueira. Logics of repeating values on data trees and branching counter systems. In *FoSSaCS '17*, volume 10203 of *Lect. Notes in Comput. Sci.*, pages 196–212. Springer, 2017b. doi:10.1007/978-3-662-54458-7_12. Cited on page 77.

L. Afanasiev and M. Marx. An analysis of XQuery benchmarks. *Inform. Sys.*, 33(2):155–181, 2008. doi:10.1016/j.is.2007.05.002. Cited on page 90.

L. Afanasiev, P. Blackburn, I. Dimitriou, B. Gaiffe, E. Goris, M. Marx, and M. de Rijke. PDL for ordered trees. *J. Appl. Non-Classical Log.*, 15(2):115–135, 2005. doi:10.3166/

jancl.15.115-135. Cited on pages 5, 7, 77, 92, and 107.

R. Alur and T. A. Henzinger. A really temporal logic. *J. ACM*, 41(1):181–203, 1994. Cited on pages 6, 8, 16, 59, 60, and 115.

D. Arroyuelo, F. Claude, S. Maneth, V. Mäkinen, G. Navarro, K. Nguyễn, J. Sirén, and N. Välimäki. Fast in-memory XPath search using compressed indexes. *Softw. Pract. & Exp.*, 45(3):399–434, 2015. doi:10.1002/spe.2227. Cited on page 3.

A. Avron. On modal systems having arithmetical interpretations. *J. Symb. Log.*, 49(3):935–942, 1984. doi:10.2307/2274147. Cited on pages 46, 58, and 115.

A. Avron. Hypersequents, logical consequence and intermediate logics for concurrency. *Annals of Mathematics and Artificial Intelligence*, 4(3–4):225–248, 1991. doi:10.1007/BF01531058. Cited on page 16.

D. Baelde, S. Lunel, and S. Schmitz. A sequent calculus for a modal logic on finite data trees. In *CSL '16*, volume 62 of *Leibniz Int. Proc. Inf.*, pages 32:1–32:16. LZI, 2016. doi:10.4230/LIPIcs.CSL.2016.32. Cited on pages 11, 16, 65, 77, and 117.

D. Baelde, A. Lick, and S. Schmitz. A hypersequent calculus with clusters for linear frames. In *AiML '18*, volume 12 of *Advances in Modal Logic*, pages 36–55. College Publications, 2018a. URL https://hal.inria.fr/hal-01756126. Cited on pages 11, 43, and 44.

D. Baelde, A. Lick, and S. Schmitz. A hypersequent calculus with clusters for tense logic over ordinals. In *FSTTCS '18*, volume 122 of *Leibniz Int. Proc. Inf.*, pages 15:1–15:19. LZI, 2018b. doi:10.4230/LIPIcs.FSTTCS.2018.15. Cited on pages 11 and 49.

D. Baelde, A. Lick, and S. Schmitz. Decidable XPath fragments in the real world. In *PODS '19*. ACM, 2019a. URL https://hal.inria.fr/hal-01852475. Cited on page 11.

D. Baelde, A. Lick, and S. Schmitz. XPath benchmark, version 2.0, 2019b. URL https://hal.inria.fr/hal-02079114. Cited on pages 11 and 78.

D. Baelde, A. Lick, and S. Schmitz. XPath parser, version 1.0, 2019c. URL https://hal.inria.fr/hal-02079276. Cited on pages 11 and 78.

B. Bednarczyk, W. Charatonik, and E. Kieronski. Extending two-variable logic on trees. In *CSL '17*, volume 82 of *Leibniz Int. Proc. Inf.* LZI, 2017. doi:10.4230/LIPIcs.CSL.2017.11. Cited on page 113.

N. D. Belnap. Display logic. *J. Philos. Logic*, 11(4):375–417, 1982. doi:10.1007/BF00284976. Cited on page 16.

S. Benaim, M. Benedikt, W. Charatonik, E. Kieroński, R. Lenhardt, F. Mazowiecki, and J. Worrell. Complexity of two-variable logic on finite trees. *ACM Trans. Comput. Logic*, 17(4):32, 2016. doi:10.1145/2996796. Cited on page 94.

M. Benedikt and J. Cheney. Destabilizers and independence of XML updates. In *VLDB '10*, volume 3, pages 906–917. VLDB Endowment, 2010. doi:10.14778/1920841.1920956. Cited on page 113.

M. Benedikt and C. Koch. XPath leashed. *ACM Comput. Surv.*, 41(1):3, 2009. doi:10.1145/1456650.1456653. Cited on pages 3, 91, 108, and 113.

M. Benedikt, W. Fan, and F. Geerts. XPath satisfiability in the presence of DTDs. *J. ACM*, 55 (2):8, 2008. doi:10.1145/1346330.1346333. Cited on pages 77 and 91.

H. Björklund and T. Schwentick. On notions of regularity for data languages. *Theor. Comput. Sci.*, 411(4-5):702–715, 2010. doi:10.1016/j.tcs.2009.10.009. Cited on page 7.

P. Blackburn, W. Meyer-Viol, and M. d. Rijke. A proof system for finite trees. In *CSL '95*, volume 1092 of *Lect. Notes in Comput. Sci.*, pages 86–105. Springer, 1996. doi:10.1007/3-540-61377-3_33. Cited on page 113.

P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2001. Cited on pages 15, 20, 21, 23, 26, 27, 30, 33, 40, 43, 45, 105, and 115.

M. Bojańczyk and P. Parys. XPath evaluation in linear time. *J. ACM*, 58(4):17, 2011. doi:10.1145/1989727.1989731. Cited on page 3.

M. Bojańczyk, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data trees and XML reasoning. *J. ACM*, 56(3):13, 2009. doi:10.1145/1516512.1516515. Cited on pages 5, 10, 77, 78, and 94.

M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Logic*, 12(4):27:1–27:26, 2011. doi:10.1145/1970398.1970403. Cited on pages 6, 7, 9, 59, 60, 71, 72, and 116.

B. Bollig. An automaton over data words that captures EMSO logic. In *Concur '11*, volume 6901 of *Lect. Notes in Comput. Sci.*, pages 171–186. Springer, 2011. doi:10.1007/978-3-642-23217-6_12. Cited on pages 2 and 59.

A. Bonifati, W. Martens, and T. Timm. An analytical study of large SPARQL query logs. In *VLDB '17*, volume 11, pages 149–161. VLDB Endowment, 2017. doi:10.14778/3149193.3149196. Cited on page 90.

K. Brünnler. Deep sequent systems for modal logic. *Arch. Math. Logic*, 48(6):551–577, 2009. doi:10.1007/s00153-009-0137-3. Cited on page 16.

V. Bruyère and O. Carton. Automata on linear orderings. *J. Comput. Syst. Sci.*, 73(1):1–24, Feb. 2007. doi:10.1016/j.jcss.2006.10.009. Cited on page 15.

D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. An automata-theoretic approach to Regular XPath. In *DBPL '09*, volume 5708 of *Lect. Notes in Comput. Sci.*, pages 18–35. Springer, 2009. doi:10.1007/978-3-642-03793-1_2. Cited on page 7.

B. ten Cate. The expressivity of XPath with transitive closure. In *PODS '06*, pages 328–337. ACM, 2006. doi:10.1145/1142351.1142398. Cited on page 107.

B. ten Cate and C. Lutz. The complexity of query containment in expressive fragments of XPath 2.0. *J. ACM*, 56(6):31:1–31:48, 2009. doi:10.1145/1568318.1568321. Cited on pages 7, 10, 77, 78, 82, 91, and 93.

B. ten Cate and L. Segoufin. XPath, transitive closure logic, and nested tree walking automata. In *PODS '08*, pages 251–260. ACM, 2008. doi:10.1145/1376916.1376952. Cited on page 7.

J. Cheney. Satisfiability algorithms for conjunctive queries over trees. In *ICDT '11*, pages 150–161. ACM, 2011. doi:10.1145/1938551.1938572. Cited on page 113.

E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logic of Programs*, volume 131 of *Lect. Notes in Comput. Sci.*, pages 52–71. Springer, 1981. doi:10.1007/BFb0025774. Cited on page 19.

N. B. Cocchiarella. *Tense and Modal Logic: a Study in the Topology of Temporal Reference*. PhD thesis, University of California, Los Angeles, 1965. Cited on pages 15, 21, and 45.

E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6): 377–387, 1970. doi:10.1145/362384.362685. Cited on page 1.

T. Colcombet and A. Manuel. Generalized data automata and fixpoint logic. In *FSTTCS '14*, volume 29 of *Leibniz Int. Proc. Inf.*, pages 267–278. LZI, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.267. Cited on pages 7, 9, and 59.

S. A. Cook. The complexity of theorem-proving procedures. In *STOC '71*, pages 151–158. ACM, 1971. doi:10.1145/800157.805047. Cited on page 17.

J. Cristau. Automata and temporal logic over arbitrary linear time. In *FSTTCS '09*, volume 4 of *Leibniz Int. Proc. Inf.*, pages 133–144. LZI, 2009. doi:10.4230/LIPIcs.FSTTCS.2009.2313. Cited on page 45.

W. Czerwinski, C. David, F. Murlak, and P. Parys. Reasoning about integrity constraints for tree-structured data. *Theor. Comput. Syst.*, 62(4):941–976, 2017. doi:10.1007/s00224-017-9771-z. Cited on pages 10, 78, and 95.

W. Czerwiński, S. Lasota, R. Lazić, J. Leroux, and F. Mazowiecki. The reachability problem for Petri nets is not elementary. In *STOC '19*. ACM, 2019. To appear. Cited on pages 8 and 72.

A. Das and D. Pous. A cut-free cyclic proof system for Kleene algebra. In *TABLEAUX '17*, volume 10501 of *Lect. Notes in Comput. Sci.*, pages 261–277. Springer, 2017. doi:10.1007/978-3-319-66902-1_16. Cited on page 16.

S. Demri and R. Lazić. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Logic*, 10(3):16, 2009. doi:10.1145/1507244.1507246. Cited on pages 6, 7, 8, 11, 16, 59, 60, 64, and 115.

S. Demri and D. Nowak. Reasoning about transfinite sequences. *Int. J. Fund. Comput. Sci.*, 18 (01):87–112, 2007. doi:10.1142/S0129054107004589. Cited on page 2.

S. Demri and A. Rabinovich. The complexity of linear-time temporal logic over the class of ordinals. *Logic. Meth. in Comput. Sci.*, 6(4):9, 2010. doi:10.2168/LMCS-6(4:9)2010. Cited on pages 15, 45, 46, and 56.

S. Demri, D. D'Souza, and R. Gascon. Temporal logics of repeating values. *J. Logic Comput.*, 22(5):1059–1096, 2012. Cited on pages 6 and 59.

S. Demri, D. Figueira, and M. Praveen. Reasoning about data repetitions with counter systems. *Logic. Meth. in Comput. Sci.*, 12(3):1, 2016. doi:10.2168/LMCS-12(3:1)2016. Cited on pages 6, 8, 59, and 72.

K. Etessami, M. Y. Vardi, and T. Wilke. First-order logic with two variables and unary temporal logic. *Inform. and Comput.*, 179(2):279–295, 2002. doi:10.1006/inco.2001.2953. Cited on pages 9, 24, 30, 45, 58, 72, and 116.

M. Farooque and S. Graham-Lengrand. Sequent calculi with procedure calls. Technical report, INRIA, 2013. URL http://arxiv.org/abs/1304.6279. Cited on page 117.

D. Figueira. Decidability of downward XPath. *ACM Trans. Comput. Logic*, 13(4):34, 2012a. doi:10.1145/2362355.2362362. Cited on pages 7, 9, 59, 77, 78, 91, 94, and 107.

D. Figueira. Alternating register automata on finite words and trees. *Logic. Meth. in Comput. Sci.*, 8(1):22, 2012b. doi:10.2168/LMCS-8(1:22)2012. Cited on pages 7, 9, 59, 77, 78, 91, 94, and 107.

D. Figueira. On XPath with transitive axes and data tests. In *PODS '13*, pages 249–260. ACM, 2013. doi:10.1145/2463664.2463675. Cited on pages 77 and 91.

D. Figueira. Automata column: Satisfiability of XPath on data trees. *SIGLOG News*, 5(2):4–16, 2018. doi:10.1145/3212019.3212021. Cited on page 94.

D. Figueira and L. Segoufin. Future-looking logics on data words and trees. In *MFCS '09*, volume 5734 of *Lect. Notes in Comput. Sci.*, pages 331–343. Springer, 2009. doi:10.1007/978-3-642-03816-7_29. Cited on pages 6, 7, 9, 10, 59, 60, 64, 94, 98, 100, 107, and 116.

D. Figueira and L. Segoufin. Bottom-up automata on data trees and vertical XPath. *Logic. Meth. in Comput. Sci.*, 13(4):5, 2017. doi:10.23638/LMCS-13(4:5)2017. Cited on pages 7, 10, 59, 77, 78, 91, 94, and 107.

M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2):194–211, 1979. doi:10.1016/0022-0000(79)90046-1. Cited on page 59.

M. Franceschet. XPathMark – an XPath benchmark for XMark generated data. In *XSYM '05*, volume 3671 of *Lect. Notes in Comput. Sci.*, pages 129–143, 2005. doi:10.1007/11547273_10. Cited on page 90.

O. Gauwin, J. Niehren, and S. Tison. Queries on XML streams with bounded delay and concurrency. *Inform. and Comput.*, 209(3):409–442, 2011. doi:10.1016/j.ic.2010.08.003. Cited on pages 2 and 59.

F. Geerts and W. Fan. Satisfiability of XPath queries with sibling axes. In *DBPL '05*, pages 122–137. Springer, 2005. doi:10.1007/11601524_8. Cited on pages 10, 77, 78, 91, 93, and 94.

P. Genevès and J.-Y. Vion-Dury. Logic-based XPath optimization. In *DocEng '04*, pages 211–219. ACM, 2004. doi:10.1145/1030397.1030437. Cited on page 4.

P. Genevès, N. Layaïda, A. Schmitt, and N. Gesbert. Efficiently deciding μ-calculus with converse over finite trees. *ACM Trans. Comput. Logic*, 16(2):16, 2015. doi:10.1145/2724712. Cited on page 93.

G. Gentzen. Untersuchungen über das logische Schließen. I. *Math. Z.*, 39(1):176–210, 1935. doi:10.1007/BF01201353. Cited on pages 10 and 16.

P. Godefroid and P. Wolper. A partial approach to model checking. *Inform. and Comput.*, 110 (2):305–326, 1994. doi:10.1006/inco.1994.1035. Cited on page 2.

G. Gottlob and C. Koch. Monadic queries over tree-structured data. In *LICS '02*, pages 189–202, 2002. doi:10.1109/LICS.2002.1029828. Cited on pages 78, 91, and 92.

G. Gottlob, C. Koch, R. Pichler, and L. Segoufin. The complexity of XPath query evaluation and XML typing. *J. ACM*, 52(2):284–335, 2005. doi:10.1145/1059513.1059520. Cited on page 3.

J. Goubault-Larrecq and J. Olivain. On the efficiency of mathematics in intrusion detection: The NetEntropy case. In *FPS '14*, volume 8352 of *Lect. Notes in Comput. Sci.*, pages 3–16. Springer, 2014. doi:10.1007/978-3-319-05302-8_1. Cited on page 2.

S. Graham-Lengrand. Psyche: A proof-search engine based on sequent calculus with an LCF-style architecture. In *TABLEAUX '13*, volume 8123 of *Lect. Notes in Comput. Sci.*, pages 149–156. Springer, 2013. doi:10.1007/978-3-642-40537-2_14. Cited on page 117.

R. Grigore, D. Distefano, R. L. Petersen, and N. Tzevelekos. Runtime verification based on register automata. In *TACAS '13*, volume 7795 of *Lect. Notes in Comput. Sci.*, pages 260–276. Springer, 2013. doi:10.1007/978-3-642-36742-7_19. Cited on pages 2 and 59.

J. Groppe and S. Groppe. A prototype of a schema-based XPath satisfiability tester. In *DEXA '06*, volume 4080 of *Lect. Notes in Comput. Sci.*, pages 93–103. Springer, 2006. doi:10.1007/11827405_10. Cited on page 4.

J. Hidders. Satisfiability of XPath expressions. In *DBPL '03*, volume 2921 of *Lect. Notes in Comput. Sci.*, pages 21–36. Springer, 2004. doi:10.1007/978-3-540-24607-7_3. Cited on pages 78, 91, 99, and 100.

A. Indrzejczak. Cut-free hypersequent calculus for S4.3. *Bull. Sec. Logic*, 41(1/2):89–104, 2012. Cited on page 16.

A. Indrzejczak. Eliminability of cut in hypersequent calculi for some modal logics of linear frames. *Inf. Process. Lett.*, 115(2):75–81, 2015. doi:10.1016/j.ipl.2014.07.002. Cited on page 16.

A. Indrzejczak. Linear time in hypersequent framework. *Bull. Symb. Log.*, 22(1):121–144, 2016. doi:10.1017/bsl.2016.2. Cited on pages 10, 16, 21, 23, 25, 26, 29, 30, 31, 40, 41, 42, 58, 72, and 115.

A. Indrzejczak. Cut elimination theorem for non-commutative hypersequent calculus. *Bull. Sec. Logic*, 46(1/2):135–149, 2017. doi:10.18778/0138-0680.46.1.2.10. Cited on pages 16, 22, and 26.

F. Jacquemard, L. Segoufin, and J. Dimino. $\mathrm{FO}^2(<, +1, \sim)$ on data trees, data tree automata and branching vector addition systems. *Logic. Meth. in Comput. Sci.*, 12(2):3, 2016. doi:10.2168/LMCS-12(2:3)2016. Cited on page 94.

M. Jurdziński and R. Lazić. Alternation-free modal mu-calculus for data trees. In *LICS '07*, pages 131–140. IEEE, 2007. doi:10.1109/LICS.2007.11. Cited on page 10.

M. Jurdziński and R. Lazić. Alternating automata on data trees and XPath satisfiability. *ACM Trans. Comput. Logic*, 12(3):19, 2011. doi:10.1145/1929954.1929956. Cited on pages 77 and 91.

M. Kaminski and N. Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994. doi:10.1016/0304-3975(94)90242-9. Cited on page 7.

M. Kanovich. The multiplicative fragment of linear logic is NP-complete. Technical Report X-91-13, Institute for Language, Logic, and Information, 1991. Cited on page 16.

A. Kara, T. Schwentick, and T. Zeume. Temporal logics on words with multiple data values. In *FSTTCS '10*, volume 8 of *Leibniz Int. Proc. Inf.*, pages 481–492. LZI, 2010. doi:10.4230/LIPIcs.FSTTCS.2010.481. Cited on page 9.

R. Kashima. Cut-free sequent calculi for some tense logics. *Stud. Logica*, 53(1):119–135, 1994. doi:10.1007/BF01053026. Cited on page 16.

A. Kiezun, V. Ganesh, S. Artzi, P. J. Guo, P. Hooimeijer, and M. D. Ernst. HAMPI: A solver for word equations over strings, regular expressions, and context-free grammars. *ACM Trans. Softw. Eng. Methodol.*, 21(4), 2013. doi:10.1145/2377656.2377662. Cited on page 113.

M. Kracht. Power and weakness of the modal display calculus. In *Proof Theory of Modal Logic*, pages 93–121. Springer, 1996. doi:10.1007/978-94-017-2798-3_7. Cited on page 16.

H. Kurokawa. Hypersequent calculi for modal logics extending S4. In *JSAI-isAI '13*, volume 8417 of *Lect. Notes in Comput. Sci.*, pages 51–68. Springer, 2014. doi:10.1007/978-3-319-10061-6_4. Cited on page 16.

F. Laroussinie, N. Markey, and Ph. Schnoebelen. Temporal logic with forgettable past. In *LICS '02*, 2002. doi:10.1109/LICS.2002.1029846. Cited on pages 21 and 45.

R. Lazić. Safety alternating automata on data words. *ACM Trans. Comput. Logic*, 12(2):10:1–10:24, 2011. doi:10.1145/1877714.1877716. Cited on pages 6, 7, and 59.

R. Lazić and S. Schmitz. Non-elementary complexities for branching VASS, MELL, and extensions. *ACM Trans. Comput. Logic*, 16(3), 2015. doi:10.1145/2733375. Cited on page 94.

B. Lellmann. Linear nested sequents, 2-sequents and hypersequents. In *TABLEAUX '15*, volume 9323 of *Lect. Notes in Comput. Sci.*, pages 135–150. Springer, 2015. doi:10.1007/978-3-319-24312-2_10. Cited on pages 16 and 25.

J. Leroux and S. Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *LICS '19*. IEEE, 2019. URL http://arxiv.org/abs/1903.08575. To appear. Cited on pages 8 and 72.

T. Liang, A. Reynolds, N. Tsiskaridze, C. Tinelli, C. Barrett, and M. Deters. An efficient SMT solver for string constraints. *Form. Methods in Syst. Des.*, 48(3):206–234, 2016. doi:10.1007/s10703-016-0247-6. Cited on page 113.

O. Lichtenstein, A. Pnueli, and L. D. Zuck. The glory of the past. In *Workshop on Logics of Programs*, volume 193 of *Lect. Notes in Comput. Sci.*, pages 196–218. Springer, 1985. doi:10.1007/3-540-15648-8_16. Cited on pages 21 and 45.

A. Lick. Systèmes de preuves pour logiques modales. Rapport de Master, Master Parisien de Recherche en Informatique, Paris, France, Aug. 2016. URL http://www.lsv.fr/Publis/PAPERS/PDF/m2-lick.pdf. Cited on page 21.

A. Lick. A Hypersequent Calculus with Clusters for Data Logic over Ordinals. In *TABLEAUX '19*. Springer, 2019. URL https://hal.archives-ouvertes.fr/hal-02165359. Cited on page 11.

P. Lincoln, J. Mitchell, A. Scedrov, and N. Shankar. Decision problems for propositional linear logic. *Ann. Pure App. Logic*, 56(1–3):239–311, 1992. doi:10.1016/0168-0072(92)90075-B. Cited on page 16.

S. Malik and L. Zhang. Boolean satisfiability from theoretical hardness to practical success. *Commun. ACM*, 52(8):76–82, 2009. doi:10.1145/1536616.1536637. Cited on page 17.

A. Manuel and A. V. Sreejith. Two-variable logic over countable linear orderings. In *MFCS '16*, volume 58 of *Leibniz Int. Proc. Inf.*, pages 66:1–66:13. LZI, 2016. doi:10.4230/LIPIcs.MFCS.2016.66. Cited on pages 23, 30, and 43.

M. Marx. XPath and modal logics of finite DAG's. In *TABLEAUX '03*, volume 2796 of *Lect. Notes in Comput. Sci.*, pages 150–164. Springer, 2003. doi:10.1007/978-3-540-45206-5_13. Cited on page 108.

M. Marx. Conditional XPath. *ACM Trans. Database Syst.*, 30(4):929–959, 2005. doi:10.1145/1114244.1114247. Cited on pages 92 and 107.

M. Marx and M. De Rijke. Semantic characterizations of navigational XPath. *SIGMOD Rec.*, 34 (2):41–46, 2005. doi:10.1145/1083784.1083792. Cited on page 93.

S. Negri. Proof analysis in modal logic. *J. Philos. Logic*, 34(5):507–544, 2005. doi:10.1007/s10992-005-2267-3. Cited on page 16.

F. Neven and T. Schwentick. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. *Logic. Meth. in Comput. Sci.*, 2(3):1, 2006. doi:10.2168/LMCS-2(3:1)2006. Cited on pages 77, 81, and 91.

H. Ono and A. Nakamura. On the size of refutation Kripke models for some linear modal and tense logics. *Studia Logica*, 39(4):325–333, 1980. doi:10.1007/BF00713542. Cited on pages 9, 22, 26, 27, 30, 31, 41, 42, 43, 44, 45, and 115.

M. Otto. Two variable first-order logic over ordered domains. *J. Symb. Log.*, 66(2):685–702, 2001. doi:10.2307/2695037. Cited on page 58.

F. Picalausa and S. Vansummeren. What are real SPARQL queries like? In *SWIM '11*, pages 7:1–7:6. ACM, 2011. doi:10.1145/1999299.1999306. Cited on page 90.

A. Pnueli. The temporal logic of programs. In *FOCS '77*, pages 46–57. IEEE, 1977. doi:10.1109/SFCS.1977.32. Cited on pages 6, 19, 45, and 59.

F. Poggiolesi. The method of tree-hypersequents for modal propositional logic. In *Trends in Logic IV*, volume 28 of *Trends in Logic*, pages 31–51. Springer, 2009a. doi:10.1007/978-1-4020-9084-4_3. Cited on page 16.

F. Poggiolesi. A purely syntactic and cut-free sequent calculus for the modal logic of provability. *Rev. Symb. Logic*, 2(4):593–611, 2009b. doi:10.1017/S1755020309990244. Cited on page 16.

A. N. Prior. *Time and Modality*. Oxford University Press, 1957. Cited on pages 19 and 45.

A. Rabinovich. Temporal logics over linear time domains are in PSPACE. *Inform. and Comput.*, 210:40–67, 2012. doi:10.1016/j.ic.2011.11.002. Cited on page 45.

G. S. Rohde. *Alternating Automata and the Temporal Logic of Ordinals*. PhD thesis, University of Illinois at Urbana-Champaign, 1997. Cited on pages 15 and 45.

D. Rouhling, M. Farooque, S. Graham-Lengrand, A. Mahboubi, and J.-M. Notin. Axiomatic constraint systems for proof search modulo theories. In *FroCoS '15*, volume 9322 of *Lect. Notes in Comput. Sci.*, pages 220–236. Springer, 2015. doi:10.1007/978-3-319-24246-0_14.

Cited on page 117.

A. Schmidt, F. Waas, M. Kersten, M. J. Carey, I. Manolescu, and R. Busse. XMark: A benchmark for XML data management. In *VLDB '02*, pages 974–985. VLDB Endowment, 2002. Cited on page 90.

T. Schwentick. XPath query containment. *SIGMOD Rec.*, 33(1):101–109, 2004. doi:10.1145/974121.974140. Cited on page 91.

A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985. doi:10.1145/3828.3837. Cited on pages 8, 19, 21, and 45.

A. P. Sistla and L. D. Zuck. Reasoning in a restricted temporal logic. *Inform. and Comput.*, 102 (2):167–195, 1993. doi:10.1006/inco.1993.1006. Cited on page 45.

M.-T. Trinh, D.-H. Chu, and J. Jaffar. S3: A symbolic string solver for vulnerability detection in Web applications. In *CCS '14*, pages 1232–1243. ACM, 2014. doi:10.1145/2660267.2660372. Cited on page 113.

M. Y. Vardi. Reasoning about the past with two-way automata. In *ICALP '98*, volume 1443 of *Lect. Notes in Comput. Sci.*, pages 628–641. Springer, 1998. doi:10.1007/BFb0055090. Cited on pages 7 and 15.

M. Y. Vardi. Boolean satisfiability: theory and engineering. *Commun. ACM*, 57(3):5–5, 2014. doi:10.1145/2578043. Cited on page 17.

M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *LICS '86*, pages 322–331. IEEE, 1986. Cited on pages 7 and 15.

Y. Zheng, X. Zhang, and V. Ganesh. Z3-str: A Z3-based string solver for web application analysis. In *ESEC/FSE '13*, pages 114–124. ACM, 2013. doi:10.1145/2491411.2491456. Cited on page 113.

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre: Logiques de requêtes à la XPath : systèmes de preuve et pertinence pratique

Mots clés: logique, satisfiabilité, systèmes de preuve, hyperséquent, amas, ordinaux, mot de données, arbre de donnée, jeu de tests, XPath

Résumé: Motivées par de nombreuses applications allant du traitement XML à la vérification d'exécution de programmes, de nombreuses logiques sur les arbres de données et les flux de données ont été développées dans la littérature. Celles-ci offrent divers compromis entre expressivité et complexité algorithmique ; leur problème de satisfiabilité a souvent une complexité non élémentaire ou peut même être indécidable. De plus, leur étude à travers des approches de théories des modèles ou de théorie des automates peuvent être algorithmiquement impraticables ou manquer de modularité.
Dans une première partie, nous étudions l'utilisation de systèmes de preuve comme un moyen modulaire de résoudre le problème de satisfiabilité des données logiques sur des structures linéaires. Pour chaque logique considérée, nous développons un calcul d'hyperséquents correct et complet et décrivons une stratégie de recherche de preuve optimale donnant une procédure de décision NP. En particulier, nous présentons un fragment NP-complet de la logique temporelle sur les ordinaux avec données, la logique complète étant indécidable, qui est exactement aussi expressif que le fragment à deux variables de la logique du premier ordre sur les ordinaux avec données.
Dans une deuxième partie, nous menons une étude empirique des principales logiques à la XPath décidables proposées dans la littérature. Nous présentons un jeu de tests que nous avons développé à cette fin et examinons comment ces logiques pourraient être étendues pour capturer davantage de requêtes du monde réel sans affecter la complexité de leur problème de satisfiabilité. Enfin, nous analysons les résultats que nous avons recueillis à partir de notre jeu de tests et identifions les nouvelles fonctionnalités à prendre en charge afin d'accroître la couverture pratique de ces logiques.

Title: XPath-like Query Logics: Proof Systems and Real-World Applicability

Keywords: logic, satisfiability, proof system, hypersequent, cluster, ordinal, data word, data tree, benchmark, XPath

Abstract: Motivated by applications ranging from XML processing to runtime verification of programs, many logics on data trees and data streams have been developed in the literature. These offer different trade-offs between expressiveness and computational complexity; their satisfiability problem has often non-elementary complexity or is even undecidable. Moreover, their study through model-theoretic or automata-theoretic approaches can be computationally impractical or lacking modularity.
In a first part, we investigate the use of proof systems as a modular way to solve the satisfiability problem of data logics on linear structures. For each logic we consider, we develop a sound and complete hypersequent calculus and describe an optimal proof search strategy yielding an NP decision procedure. In particular, we exhibit an NP-complete fragment of the tense logic over data ordinals—the full logic being undecidable—, which is exactly as expressive as the two-variable fragment of the first-order logic on data ordinals.
In a second part, we run an empirical study of the main decidable XPath-like logics proposed in the literature. We present a benchmark we developed to that end, and examine how these logics could be extended to capture more real-world queries without impacting the complexity of their satisfiability problem. Finally, we discuss the results we gathered from our benchmark, and identify which new features should be supported in order to increase the practical coverage of these logics.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France