

Reasoning on Words and Trees with Data^{*}

Author: Diego Figueira Supervisors: Luc Segoufin, Stéphane Demri

Abstract

A data word (resp. a data tree) is a finite word (resp. tree) whose every position carries a letter from a finite alphabet and a *datum* from an infinite domain. In this thesis we investigate automata and logics for data words and data trees with decidable reasoning problems: we focus on the emptiness problem in the case of automata, and the satisfiability problem in the case of logics. These problems are pertinent to areas such as software verification, data-driven systems verification, and static analysis of database query languages.

On data words, we present a decidable extension of a model of alternating register automata studied by Demri and Lazić. Further, we investigate the satisfiability problem for a linear-time temporal logic on data words. This logic features one register for storing and comparing data values seen along the data word. The logic has been previously studied by Demri and Lazić, here we show that this logic extended with quantification over data values is decidable. In addition, we prove that even when the logic is restricted to having only a ‘future’ modality (*i.e.*, no next modality), its satisfiability problem has a non-primitive-recursive lower bound.

On data trees, we consider three decidable automata models with different expressive powers. We first introduce the Downward Data automaton (DD automata). Its execution consists in a transduction of the finite labeling of the tree, and a verification of data properties for every subtree of the transduced tree. This model is closed under boolean operations, although the tests it can make on the order of the siblings is limited. Its emptiness problem is in 2EXPTIME. On the contrary, the other two automata models we introduce have an emptiness problem with

a non-primitive-recursive complexity, and are closed under intersection and union, but not complementation. They are both alternating automata with one register to store and compare data values. We introduce an automata model that extends the alternating top-down register automata model (ATRA) studied by Jurdiński and Lazić. We exhibit similar decidable extensions as in the case of data words. This class of automata can test for any regular tree language—in contrast to DD automata. Finally, we consider a bottom-up alternating tree automaton with one register (called BUDA). Although BUDA are one-way, they can test data properties by navigating the tree in both directions: upward and downward. In opposition to the extension of ATRA, this automaton cannot test for properties on the sequence of siblings (like, for example, the order in which labels appear). In order to show decidability for these automata models we introduce some novel approaches, revealing strong connections with the theory of well-structured transition systems.

Each of these three models has a connection with the logic XPath—a logic conceived for XML documents, which can be seen as data trees. Through the aforementioned automata we show that the satisfiability of three natural fragments of XPath are decidable. These fragments are: *downward XPath*, where navigation can only be done by child and descendant axes; *forward XPath*, where navigation also contains the next sibling axis and its transitive closure; and *vertical XPath*, whose navigation consists in the child, descendant, parent and ancestor navigation axes. Whereas downward XPath is EXPTIME-complete, forward and vertical XPath have non-primitive-recursive lower bounds.

^{*} Thesis defended on December 6, 2010 at *École Normale Supérieure de Cachan*, in front of a jury composed of: Stéphane Demri (supervisor), Georg Gottlob (reviewer), Ranko Lazić (examiner), Leonid Libkin (examiner), Carsten Lutz (examiner), Thomas Schwentick (reviewer), and Luc Segoufin (supervisor).

1 Introduction

Words and trees are amongst the most studied structures in computer science. In this thesis, we focus on words and trees that can contain elements from some infinite alphabet (that we call *data values*), like for example the set of integers, or the set of words over the alphabet $\{a, b\}$. These kind of structures are relevant to many areas.

For instance, in software verification, one may need to decide statically whether a program satisfies some given specification; and the necessity of dealing with infinite alphabets can arise from different angles. For example, in the presence of concurrency, we have an unbounded number of processes running, each one with its process identification, and we must verify properties specifying the interplay between these processes. Further, procedures may take parameters as input, and they can hence exchange data from some unbounded domain. Infinite alphabets can also emerge as a consequence of the use of recursive procedure calls, communication through FIFO channels, etc.

Also, in a database context, infinite alphabets are a common occurrence. Let us dwell on static analysis tasks on XML documents and its query languages. In this context there are several pertinent problems serving static analysis. For example, there is the problem of *coherence*: is there a document in which a given query returns a non-empty result? The problem of *inclusion*: is it true that for any document, the result answered by one given query is contained in the result of another? And the problem of *equivalence*: do two given queries always return the same answers? These questions are at the core of many static analysis tasks. For example, by answering the coherence problem one can decide whether the computation of a query on a database can be avoided because the query contains a contradiction; and by answering the equivalence problem if one query can be safely replaced by a simpler one. All these queries recurrently need to specify properties concerning not only the labels of the nodes, but also the actual data contained in the attributes. In particular, they need to perform *joins*, that is, to describe when two data values ought to be equal or not.

Still in the context of databases, we can also regard logics that express data properties as specification languages. In verification of database-driven systems, we are provided with the specification of a system that interacts with a database, and we need to check whether it is possible to reach a state in which the database has some undesired property. In order to model the specification of the system as well as the property of the database—for example through an automaton or a logical formula—we typically need to take into account values from infinite domains.

Therefore, the study of formalisms to reason with words and trees that can carry elements from some infinite domain is relevant to all the aforementioned areas, and possibly more. To begin our study, first we need to fix once and for all the structure over which we intend to reason.

Data words and *data trees* are simple models that extend words and trees over finite alphabets. A data word is a word (*i.e.*, a finite sequence) where every position has a symbol from a finite alphabet (a *label*), and an element from some infinite domain (a *data value*). Similarly, a data tree is an unranked ordered finite tree, whose every node carries a label and a data value. By *unranked* we mean that every node has unboundedly many children, and by *ordered* that the children of a node are seen as an ordered list of subtrees, instead of a set or multiset. This model is in close relation to an XML document. Indeed, all the results we obtain on data trees can be translated into the XML setting.

To get familiar with these models, let us give some examples of possible *data properties* (*i.e.*, properties whose satisfaction rely on the model's data values) that one may be interested in verifying in a data word or a data tree.

Example 1. We have several processes (or execution threads of a process) running concurrently.¹ Each one of these has a process identifier (a number), and we model the history of execution of these processes. In the history, we record when the process i begins a task with an element (b, i) ,

¹This example is inspired on an example recurrently used by Thomas Schwentick.

$$\begin{aligned}
& \text{root} \rightarrow \text{category stock} \\
& \text{stock} \rightarrow (\text{id})^* \\
& \text{category} \rightarrow (\text{category})^* (\text{product})^* \text{featured?} \\
& \text{product} \rightarrow \text{name id}
\end{aligned}$$

Figure 1: A simple specification modeling the data trees of interest for our problem.

when it ended a task with (e, i) , and when it reads (r, i) and writes (w, i) to the hard disk, during the task. To make things interesting, suppose that once in a while there is a maintenance shutdown of the disk (performed by a process with letter s). A possible history may be one like this one.

$$(b, 1) (b, 2) (r, 1) (r, 2) (w, 1) (e, 1) (b, 1) (e, 2) (e, 1) (s, 3) (b, 4) (w, 4) (e, 4) (s, 3)$$

In this example we see that process 1 performs two tasks, in the first one it reads and then writes, and in the second one it does not read nor write. Interleaved with these tasks process 2 also performs a reading task. Then the maintenance process 3 shutdowns the disk, etc.

Let us describe some possible properties one could want to verify over a history like this.

- (W1) For every b there is a future e with the same data value.
- (W2) Further, we can ask that for every data value, the data word restricted to that data value belongs to $(s + (b \cdot \{r, w\}^* \cdot e))^*$.
- (W3) Every time a task starts, it will end before the shutdown. That is, for every b there exists an e with the same data value, that occurs before the next s .
- (W4) There exists a process whose first operation is a write.
- (W5) Each time a s appears, all the processes occurred so far do not reappear in the future. \square

Example 2. Suppose now we have a data tree containing all the products that are being sold by a website. Suppose that the shape and labels of the tree are specified with the DTD of Figure 1. A DTD is basically a set of rules of the kind “if a node has label l , then the children have certain labels”, which are read in the expected manner. For example, the root has two nodes, one with label **category** and one labeled **stock**. Each node labeled **category** has a sequence of nodes labeled **category**, a sequence of nodes labeled **product**, and finally perhaps a node labeled **featured**. The idea is that the products are organized in a hierarchy of categories (described by the **category** child of the root), and also we have under **stock** a set of product identifiers that are on stock. Further, some categories may be “featured categories” of products, obtaining some special visibility in the site, for all the products in the category and recursively in all subcategories. An example of a possible tree is shown in Figure 2.

Given a data tree that models such scenario, there might be a number of properties that we want to verify. In the spirit of depicting different sorts of properties, consider the following.

- (T1) It cannot happen that a category has the same name as a product.
- (T2) All the product id’s under the **category** subtree are different. Also, all the product id’s under the **stock** node are different.
- (T3) The name of a product allows to identify a product inside a category. In other words, all the product name in a category are different.
- (T4) Every product in stock is categorized under some category.
- (T5) All the categories have different names.

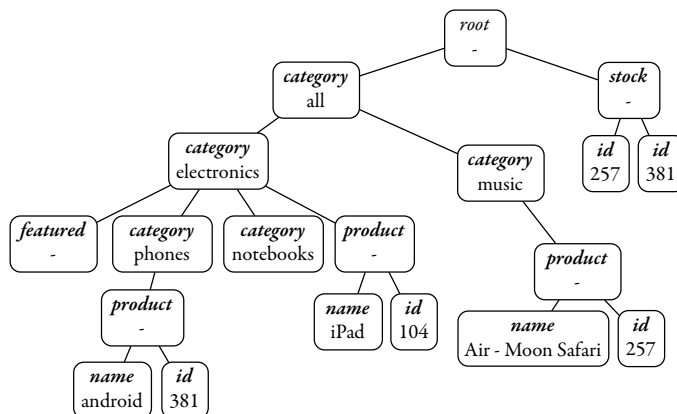


Figure 2: An example of a data tree with information about the products sold by a web site.

- (T6) There is at least one product in stock that is in some descendant category of a featured category.
- (T7) The same product cannot be in two categories in the ancestor-descendant relation. \square

These examples intend to show the variety of properties we may need to verify in an environment with data. We consider two sorts of formalisms for specifying these properties: either by means of *logics*, or by means of accepting runs of *automata*. However, it can be shown that there is no decidable logic or class of automata that is able to express all these desirable properties and is closed under conjunction or intersection, neither in the case of trees nor in the case of words. At the same time, most of these independent properties can be expressed by some decidable formalism in the literature closed under intersection. The general outlook is that there are not many expressive formalisms that are decidable, and there is also a need for more general techniques to allow to treat data values to prove decidability results.

The thesis explores expressive logics and automata with decidable reasoning tasks. We give several new results on decidable logics on these data models, and we introduce new decidable automata models. The intention of this work is to devise new approaches and techniques to work with data values, and to give new insights on the limits of what is decidable on these models.

Preliminaries

We fix some basic notation. We define $\mathbb{N} = \{1, 2, 3, \dots\}$, and $[n] = \{1, \dots, n\}$. We fix \mathbb{D} to be any infinite domain of data values; in our examples we consider $\mathbb{D} = \mathbb{N}$. In general we use letters \mathbb{A} , \mathbb{B} for finite alphabets, the letter \mathbb{D} for an infinite alphabet and the letters \mathbb{E} and \mathbb{F} for any kind of alphabet. By \mathbb{E}^* we denote the set of finite sequences over \mathbb{E} , by \mathbb{E}^+ the set of finite sequences with at least one element over \mathbb{E} . We use ‘.’ as the concatenation operator between sequences.

2 Related work

We mention briefly some works closely related to the thesis. For a broader and more detailed discussion we refer the reader to Sections 1.3, 3.2, 4.4, 5.1.1, 6.1 and 7.1.1 of the thesis.

Register Automata Kaminski and Francez (1994) introduce the class of register automata on data words, its expressiveness studied by Neven, Schwentick, and Vianu (2004). This model was extended to data trees by Kaminski and Tan (2008). The model of one-way alternating register

automata ARA is an extension studied by Demri and Lazić (2009). It has also been extended to trees, as an alternating top-down tree register automata (ATRA) by Jurdziński and Lazić (2008). Bouyer, Petit, and Thérien (2003) studied a class of automata, inspired by timed languages, coining the term *data word*. In fact, automata for timed languages and register automata are very closely related, since most problems in one model can be reduced into similar problems in the other (Figueira, Hofman, and Lasota, 2010). We extend the models ARA and ATRA while preserving decidability of the emptiness problem.

Linear temporal logics The logic for data-words most relevant to this thesis is the temporal logic LTL extended with one register for storing and comparing data values. It is denoted by LTL^\downarrow , and it was studied in (Demri and Lazić, 2009). We investigate decidable extensions and lower bounds of restrictions of this logic.

First-order logic The satisfiability of first-order logic with two variables and data equality tests is explored by Bojańczyk, Muscholl, Schwentick, Segoufin, and David (2006). The logic $FO^2(\sim, <, +1)$ is shown to have a decidable satisfiability problem. This is first-order logic where we have only two variables x and y (that we can however reuse), and three binary relations: $<$ is interpreted as the order of positions in the word, $+1$ denotes the relation of consecutive positions, and \sim relates two positions that have the same data value. This logic can equivalently be seen as an automaton called *data automaton*, or as a *class memory automata* (Björklund and Schwentick, 2010). However, these formalisms are incomparable in terms of expressiveness with respect to the automata and logics we study in this thesis. For example, data automata can express property (W2) that cannot be expressed by any of the formalisms of the thesis, but they cannot express (W3). This model was extended by Björklund and Bojańczyk (2007) to work on bounded-depth unranked data trees. If we consider that the domain is linearly-ordered, finite satisfiability for $FO^2(<, +1, \sim, <)$ becomes undecidable (Bojańczyk et al., 2006). However, Schwentick and Zeume (2010) show that it becomes decidable if we disallow the $+1$ operator. Bojańczyk and Lasota (2010) introduce a simple and powerful —undecidable— automata model called *class automata* on data trees that captures $FO^2(\sim, <, +1)$, data automata, XPath, ATRA, and some other models.

XPath Benedikt, Fan, and Geerts (2008) studied the satisfiability problem for many XPath fragments containing downwards and upwards axes, but no horizontal axes. In (Geerts and Fan, 2005), several XPath fragments with horizontal axes are treated. In this thesis we continue this study exhibiting large decidable fragments of XPath.

Bojańczyk et al. (2009) shows the decidability of $XPath^\varepsilon(\uparrow, \downarrow, \leftarrow, \rightarrow, =)$ with sibling and upward axes but restricted to local elements accessible by a “one step” relation, and to data formulas of the kind $\langle \varepsilon = \alpha \rangle$ (or \neq). We, on the other hand, work with fragments that can perform tests of nodes at possibly distant positions of the tree.

Finally, Jurdziński and Lazić (2008) shows decidability of $XPath^\varepsilon(\downarrow, \downarrow^*, \rightarrow, \rightarrow^*, =)$, a fragment of with forward axes (*i.e.*, downwards and rightwards) and where the data tests are restricted as before. This is done by translating formulas of this fragment into ATRA automata. In our work we show that in fact the full, unrestricted, forward fragment of XPath is decidable.

Well-structured transition systems We make use of techniques involving well-structured transition systems. The theory of well-structured transition systems (WSTS) is a development born in the field of verification of infinite state systems. Paraphrasing Finkel and Schnoebelen (2001), these are transition systems where the existence of a well-quasi-ordering over the infinite set of states ensures the termination of several algorithmic methods. WSTS’s are an abstract generalization of several specific structures and they allow decidability results in many fields.

3 Contribution

The contribution of this thesis is divided into two parts: data words and data trees. We make this distinction since in general we cannot always view a word as a special case of a tree. For example, notice that a move to the left followed by a move to the right in a word goes back to the same position on the word, while not necessarily in the tree case (a move to the parent followed by a move to a child may end up in a sibling node). In fact, both have useful applications, for instance one can model the traces of runs of processes, the other can model a database of products.

In both parts we study automata models and logical formalisms. We use the automata models for showing decidability of the logical formalisms we present, but the automata models are useful on their own. In fact, the automata models are more powerful than the corresponding logics we study, and we believe that they can also have applications on verification of database systems, consistency of specifications, etc.

3.1 Data words

We consider a finite word over \mathbb{E} as a function $\mathbf{w} : [n] \rightarrow \mathbb{E}$ for some $n \in \mathbb{N}$, and we define the set of words as $Words(\mathbb{E}) := \{\mathbf{w} : [n] \rightarrow \mathbb{E} \mid n \in \mathbb{N}\}$. We write $\text{pos}(\mathbf{w}) = \{1, \dots, n\}$ to denote the set of *positions* (that is, the domain of \mathbf{w}). Given $\mathbf{w} \in Words(\mathbb{E})$ and $\mathbf{w}' \in Words(\mathbb{F})$ with $\text{pos}(\mathbf{w}) = \text{pos}(\mathbf{w}') = P$, we write $\mathbf{w} \otimes \mathbf{w}' \in Words(\mathbb{E} \times \mathbb{F})$ for the word such that $\text{pos}(\mathbf{w} \otimes \mathbf{w}') = P$ and $(\mathbf{w} \otimes \mathbf{w}')(x) = (\mathbf{w}(x), \mathbf{w}'(x))$. A **data word** is a word from $Words(\mathbb{A} \times \mathbb{D})$, where \mathbb{A} is a finite alphabet of letters and \mathbb{D} is an infinite domain of *data values*.

We investigate alternating register automata and temporal logics for data words. This corresponds to Chapter 3 in the thesis, and is also included in (Figueira and Segoufin, 2009; Figueira, 2012a). We define a one-way alternating automata model over data words, with one register for storing data and comparing them for equality, and two other operators that add more expressive power, extending existing results of Demri and Lazić (2009).

Our work on register automata aims at two objectives: (1) simplifying the existent decidability proofs for the emptiness problem for alternating register automata; and (2) exhibiting decidable extensions for these models.

From the logical perspective, we work with the temporal logic LTL extended with one register for storing and comparing data values (denoted by LTL^\downarrow). This logic contains a ‘freeze’ operator to store the current datum and a ‘test’ operator to test the current datum against the stored one. We show that (a) in the case of data words, satisfiability of LTL^\downarrow with *quantification* over data values is decidable; and (b) the satisfiability problem for very weak fragments of LTL^\downarrow with one register are non-primitive-recursive—markedly, these fragments have no “next element” X modality.

3.1.1 Alternating register automata

On data words, we focus on automata with one register and alternating control called ARA (for Alternating Register Automata). ARA are one-way automata with alternating control and one register to store data values for later comparison. This automata class is known to have a decidable emptiness problem (Demri and Lazić, 2009) through a non-trivial reduction to the emptiness problem for Incrementing Counter Automata². However, the emptiness problem for ARA cannot be decided by any algorithm whose time or space is bounded by any primitive-recursive function—in this case, we say that it has non-primitive-recursive complexity.³

²An Incrementing Counter Automaton is a Minsky Counter Automaton whose runs may contain ‘errors’ that increase one or more counters non-deterministically throughout the run. This makes its emptiness problem decidable.

³More precisely, this problem sits in the class \mathfrak{F}_ω of the Fast Growing Hierarchy (Löb and Wainer, 1970)—an extension of the Grzegorzczuk Hierarchy for non-primitive-recursive functions—by a reduction to the emptiness problem for Incrementing Counter Automata (Demri and Lazić, 2009), which is in \mathfrak{F}_ω (Figueira et al., 2011, §7.2).

We extend these automata with operators that add expressive power, while preserving decidability. These operators are used to transfer the decidability to the satisfiability problem for a logic. In our work, decidability of these models is shown by interpreting the semantics of the automaton in the theory of well-structured transition systems (WSTS) (see Finkel and Schnoebelen, 2001). The object of this alternative proof is twofold. On the one hand, we propose a direct, unified and self-contained proof of the main decidability results of Demri and Lazić. Whereas in (Demri and Lazić, 2009) decidability results are shown by reduction to a class of faulty counter automata, here we avoid such translation, and we provide a simple decidability argument directly interpreting the configurations of the automata in the theory of well-structured transition systems. That is, we avoid an intermediary reduction to a faulty counter system. We stress, however, that the underlying techniques used here are somewhat similar to those used to prove decidability for Incrementing Counter Automata. On the other hand, we further generalize these results. Our proof can be easily extended to show the decidability of the nonemptiness problem for two extensions. These extensions consist in the following abilities:

- (a) the automaton can nondeterministically *guess* any data value of the domain and store it in the register; and
- (b) it can make a universal quantification over all the data values seen along the run of the automaton, and in particular over the data values seen so far.

We name these extensions **guess** and **spread** respectively. These extensions can be added to the ARA model preserving decidability, although losing closure under complementation.⁴ We call the model of alternating tree register automata with these extensions by $\text{ARA}(\text{guess}, \text{spread})$. We show that any of these extensions add expressive power. In fact, we can now express (W4) or (W5), that were not possible to express before.

We now give more details on the automata model. An alternating register automaton of $\text{ARA}(\text{spread}, \text{guess})$ consists of a finite alphabet, a finite set of states, an initial state, and a transition function that associates to each state one of the following formulas: a , \bar{a} , $\triangleright?$, $\bar{\triangleright}?$, $\text{store}(q)$, eq , $\bar{\text{eq}}$, $q \wedge q'$, $q \vee q'$, $\triangleright q$, $\text{guess}(q)$, $\text{spread}(q, q')$; where a is a letter from the alphabet and q, q' are states.

Here, a and \bar{a} are to test that the current element of the data word has (or has not) the letter a ; $\triangleright?$ and $\bar{\triangleright}?$ are to test if we are at the last element of the word (or not); \triangleright is to move to the next position to the right on the data word; $\text{store}(q)$ stores the current datum in the register and continues the execution with state q ; and eq and $\bar{\text{eq}}$ test that the current node's value is (or is not) equal to the stored. As usual, \wedge and \vee are used for alternation and nondeterminism. This formalism without the **guess** and **spread** transitions is equivalent to the ARA model of (Demri and Lazić, 2009) on finite data words. It can express properties like, *e.g.*, (W3).

As this automaton is one-way, we define its semantics as a set of ‘threads’ for each node that progress synchronously. Each thread is a pair (q, d) where q is a state and d is a data value. All threads at a node move one step forward simultaneously and then perform some non-moving transitions independently. A configuration then contains a set of *threads* that must be verified. For example, if a transition $q_1 \wedge q_2$ is triggered from a thread (q, d) , then (q, d) is removed from the configuration and (q_1, d) , (q_2, d) are added. And whenever a thread (q, d) executes a successful test (say, for example, eq), the thread is removed from the configuration.

The **guess** instruction extends the model with the ability of storing *any* datum from the domain \mathbb{D} . And the **spread** instruction is an unconventional operator in the sense that it depends on the data of *all* threads in the current configuration with a certain state. Whenever $\text{spread}(q_2, q_1)$ is executed, a new thread (q_1, d) is added to the configuration for each thread (q_2, d) present in the configuration. With this operator we can code, for example, a universal quantification over all the previous data values of the data word.

⁴In fact, we show that if we add the dual operator of any of these two extensions, the automata model becomes undecidable.

The set of transitions then defines in a natural way a relation between configurations. An accepting run on a data word is then a sequence of configurations consistent with the aforementioned relation and the data word, that starts with a configuration containing only one thread in the initial state and the first data word, and ends in a configuration with no threads. Our main result here is the following.

Theorem 3.5. The emptiness problem for $\text{ARA}(\text{guess}, \text{spread})$ is decidable.

We demonstrate that these extensions are also decidable if the data domain is equipped with a linear order and the automata model is extended accordingly (§3.3.3). Our investigation on register automata also yields new results on a class of timed automata, called alternating 1-clock timed automaton (§3.3.4).⁵ These results are included in §3.3 of the thesis and in (Figueira, 2012a).

3.1.2 Linear temporal logics

On the logical side, we center our attention on linear temporal logics.

Preliminaries

The logic $\text{LTL}_n^\downarrow(\mathcal{O})$ is a logic for data words that corresponds to the extension of the Linear Temporal Logic $\text{LTL}(\mathcal{O})$ on data words, where \mathcal{O} is a subset of the usual navigation modalities, for example $\{\text{F}, \text{U}, \text{X}\}$. The logic is extended with the ability to use n different *registers* for storing a data value for later comparisons, and it was studied in (Demri and Lazić, 2009; Demri et al., 2005). The *freeze* operator $\downarrow_i \varphi$ permits to *store* the current datum in register i and continue the evaluation of the formula φ . The operator \uparrow_i *tests* whether the current data value is equal to the one stored in register i . We use the usual navigation modalities: the next (X), future (F) and until (U) temporal operators, together with its inverse counterparts (X^{-1} , F^{-1} , U^{-1}).

As it was shown by Demri and Lazić (2009), $\text{LTL}_1^\downarrow(\text{U}, \text{X})$ has a decidable satisfiability problem with non-primitive-recursive complexity. However, as soon as $n \geq 2$, satisfiability of $\text{LTL}_n^\downarrow(\text{U}, \text{X})$ becomes undecidable. We write $\text{LTL}^\downarrow(\mathcal{O})$ as short for $\text{LTL}_1^\downarrow(\mathcal{O})$. We focus on decidable upper bounds for extensions of $\text{LTL}^\downarrow(\text{X}, \text{U})$, and on lower bounds for some fragments.

We define the semantics of the logic with only one register. Fix a finite alphabet \mathbb{A} . Sentences of $\text{LTL}^\downarrow(\mathcal{O})$, where $\mathcal{O} \subseteq \{\text{X}, \text{F}, \text{U}, \text{X}^{-1}, \text{F}^{-1}, \text{U}^{-1}\}$ are defined as follows, where a is a letter from \mathbb{A} ,

$$\varphi ::= \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg \varphi \mid a \mid \uparrow \mid \downarrow \varphi \mid o \varphi . \quad (o \in \mathcal{O})$$

Table 1 shows the definition of the satisfaction relation \models . For example, in this logic we can express (W1), or a property like “for every a element there is a future b element with the same data value” as $\text{G}(\neg a \vee \downarrow (\text{F}(b \wedge \uparrow)))$. We say that φ satisfies $\mathbf{w} = \mathbf{a} \otimes \mathbf{d}$, written $\mathbf{w} \models \varphi$, if $\mathbf{w}, 1 \models^{\mathbf{d}(1)} \varphi$.

We show that $\text{LTL}^\downarrow(\text{U}, \text{X})$ can be extended with quantification over data values, extending the previous decidability results. On the other hand, we investigate the lower bounds for very weak fragments. By the lower bounds given in (Demri and Lazić, 2009), the complexity of the satisfiability problem for $\text{LTL}^\downarrow(\text{U}, \text{X})$ is non-primitive-recursive. Here we show that even $\text{LTL}^\downarrow(\text{F})$ has non-primitive-recursive complexity, and that already $\text{LTL}^\downarrow(\text{F}, \text{F}^{-1})$ is undecidable. Further, if two registers are allowed, $\text{LTL}_2^\downarrow(\text{F})$ is also undecidable.

Upper bounds Our work on $\text{ARA}(\text{guess}, \text{spread})$ yields new decidability results on the satisfiability for some extensions of the temporal logic with one register $\text{LTL}^\downarrow(\text{U}, \text{X})$. Our automata model captures an extension of this logic with quantification over data values, where we can express for any $\varphi \in \text{LTL}^\downarrow(\text{U}, \text{X})$:

⁵This is because the problems of nonemptiness, language inclusion, language equivalence and universality are equivalent—modulo an EXPTIME reduction—for timed automata and register automata over a linearly ordered data domain (Figueira et al., 2010).

$(\mathbf{w}, i) \models^d a$	iff	$\mathbf{a}(i) = a$
$(\mathbf{w}, i) \models^d \uparrow$	iff	$d = \mathbf{d}(i)$
$(\mathbf{w}, i) \models^d \downarrow \varphi$	iff	$(\mathbf{w}, i) \models^{\mathbf{d}(i)} \varphi$
$(\mathbf{w}, i) \models^d \mathbf{U}(\varphi, \psi)$	iff	for some $i \leq j \in \text{pos}(\mathbf{w})$ and for all $i \leq k < j$ we have $(\mathbf{w}, j) \models^d \varphi$, $(\mathbf{w}, k) \models^d \psi$
$(\mathbf{w}, i) \models^d \mathbf{X}\varphi$	iff	$i + 1 \in \text{pos}(\mathbf{w})$ and $(\mathbf{w}, i + 1) \models^d \varphi$

Table 1: Semantics of $\text{LTL}^\downarrow(\mathbf{X}, \mathbf{F}, \mathbf{U}, \mathbf{X}^{-1}, \mathbf{F}^{-1}, \mathbf{U}^{-1})$ for a data word $\mathbf{w} = \mathbf{a} \otimes \mathbf{d}$ and $i \in \text{pos}(\mathbf{w})$. The interpretation of \wedge , \vee and \neg is standard. As usual, we define the *future* modality as $\mathbf{F}\varphi := \mathbf{U}(\varphi, \top) \vee \varphi$, and \mathbf{U}^{-1} , \mathbf{F}^{-1} , \mathbf{X}^{-1} as the symmetric of \mathbf{U} , \mathbf{F} , \mathbf{X} .

- “for all data values in the past, φ holds” with the formula $\forall_{\leq}^\downarrow \varphi$, and
- “there exists a data value in the future where φ holds” with the formula $\exists_{\geq}^\downarrow \varphi$.

Indeed, we show that none of these two types of properties can be expressed in the formalism of Demri and Lazić. These quantifiers may be added to $\text{LTL}^\downarrow(\mathbf{U}, \mathbf{X})$ without losing decidability.

Theorem 3.31. The satisfiability problem for $\text{LTL}^\downarrow(\mathbf{U}, \mathbf{X})$ extended with $\exists_{\geq}^\downarrow$ and $\forall_{\leq}^\downarrow$ (occurring as positive appearances) is decidable.

What is more, we show that decidability is preserved if the data domain is equipped with a linear order accessible by the logic. However, adding the *dual* of any of these operators results in an undecidable logic. These results are included in §3.5 of the thesis as well as in (Figueira, 2012a).

Lower bounds Our second contribution on these logics is on lower bounds. Thanks to Demri and Lazić we know that $\text{LTL}^\downarrow(\mathbf{F}, \mathbf{X})$ has a non-primitive-recursive lower bound, and that $\text{LTL}^\downarrow(\mathbf{F}, \mathbf{F}^{-1}, \mathbf{X})$ is undecidable. We show that these results carry over even in the absence of \mathbf{X} .

Theorem 3.38. The satisfiability problem for $\text{LTL}^\downarrow(\mathbf{F})$ is non-primitive-recursive; the satisfiability problem for $\text{LTL}^\downarrow(\mathbf{F}, \mathbf{F}^{-1})$ is undecidable.⁶

Theorem 3.39. The satisfiability for $\text{LTL}_2^\downarrow(\mathbf{F})$ is undecidable.⁶

These results are joint work with Luc Segoufin and are included in §3.6 of the thesis as well as in (Figueira and Segoufin, 2011).

3.2 Data trees

We define $\text{Trees}(\mathbb{E})$, the set of finite, ordered and unranked trees over an alphabet \mathbb{E} . A *position* in the context of a tree is an element of \mathbb{N}^* . The root’s position is the empty string and we note it ‘ ϵ ’. The position of any other node in the tree is the concatenation of the position of its parent and the node’s index in the ordered list of siblings. Thus, for example $x \cdot i$ —where $x \in \mathbb{N}^*$ and $i \in \mathbb{N}$ —is a position which is not the root, that has x as parent position, and such that there are $i - 1$ siblings to the left of $x \cdot i$. The set of trees over \mathbb{E} , noted $\text{Trees}(\mathbb{E})$, is the set of labeling functions $\mathbf{t} : P \rightarrow \mathbb{E}$, where P is a set of tree positions.⁷ We informally refer by ‘node’ to a position x of \mathbf{t} together with the value $\mathbf{t}(x)$. Given two trees $\mathbf{t}_1 \in \text{Trees}(\mathbb{E})$, $\mathbf{t}_2 \in \text{Trees}(\mathbb{F})$ with the same set of positions P , we define $\mathbf{t}_1 \otimes \mathbf{t}_2 : P \rightarrow (\mathbb{E} \times \mathbb{F})$ as $(\mathbf{t}_1 \otimes \mathbf{t}_2)(x) = (\mathbf{t}_1(x), \mathbf{t}_2(x))$. The set of **data trees** over a finite alphabet \mathbb{A} and an infinite domain \mathbb{D} is defined as $\text{Trees}(\mathbb{A} \times \mathbb{D})$. Note that every tree $\mathbf{t} \in \text{Trees}(\mathbb{A} \times \mathbb{D})$ can be decomposed into two trees $\mathbf{a} \in \text{Trees}(\mathbb{A})$ and $\mathbf{d} \in \text{Trees}(\mathbb{D})$ such that $\mathbf{t} = \mathbf{a} \otimes \mathbf{d}$. Figure 3 shows an example of a data tree.

⁶These lower bounds hold both when \mathbf{F} is considered a strict or a non-strict (reflexive) modality.

⁷A set of tree positions is any finite subset of \mathbb{N}_0^* closed under prefix, such that if $x \cdot (i + 1)$ is in the set, so is $x \cdot i$.

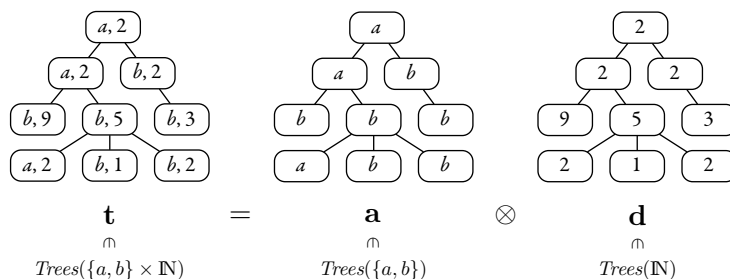


Figure 3: A data tree.

In the case of data trees we consider a widely used logic for trees with data called **XPath**. Although the satisfiability problem for this logic is undecidable, we exhibit three large fragments—called downward, forward, and vertical **XPath**—that are decidable. The first one is in **EXPTIME** and the two others have non-primitive-recursive complexity (these bounds are optimal). The decidability proofs are in each case based in a reduction to the emptiness problem for one of the automata models we introduce.

From the automata side, we consider three decidable automata models with different characteristics that allow to express different kinds of data properties. All these three models of automata have a decidable emptiness problem, and they all have connections with the logic **XPath**. Our work also gives alternative, arguably simpler, decidability proofs for the known results from (Jurdziński and Lazić, 2008). This simplification allows us to extend the decidability results with operators that add expressive power (as in the case for data words).

3.2.1 Preliminaries: XPath

XPath is a logic for XML documents (which are essentially data trees). Expressions of this logic can navigate the tree by composing binary relations from a set of basic relations, that can contain the parent relation (noted \uparrow), child (\downarrow), ancestor (\uparrow^*), descendant (\downarrow_*), next sibling to the right (\rightarrow) or to the left (\leftarrow), and their transitive closures (\rightarrow^* , $*\leftarrow$). For example “ $\uparrow[a]\uparrow\downarrow[b]$ ” defines the relation between two nodes x, y such that y is an uncle of x labeled b and the parent of x is labeled a . Boolean tests are built by using these compound relations. An expression like $\langle \alpha \rangle$ (for α a relation) tests that there exists a node accessible with the relation α from the current node. Most importantly, a data test like $\langle \alpha = \beta \rangle$ (resp. $\langle \alpha \neq \beta \rangle$) tests that there are two nodes reachable from the current node with the relations α and β that have the same (resp. different) data value. We consider three natural fragments of **XPath**, according to which set of basic relations (usually called axes) we use: downward \downarrow, \downarrow_* ; forward $\downarrow, \downarrow_*, \rightarrow, \rightarrow^*$; or vertical $\downarrow, \downarrow_*, \uparrow, \uparrow^*$. We call these fragments respectively the downward, forward and vertical fragments.

We define **XPath** over data trees. We prefer to define and work with data trees because it is a simpler model. In the thesis we show that every result obtained for data trees holds also for XML documents (§4.3, §5.4.3, §6.4.4, §7.4.1).⁸

XPath is arguably the most widely used XML query language. It is implemented in XSLT and XQuery and it is used as a constituent part of several specification and update languages. **XPath** is fundamentally a general purpose language for addressing, searching, and matching pieces of an XML document. It is an open standard and constitutes a World Wide Web Consortium (W3C) Recommendation (Clark and DeRose, 1999), implemented in most languages and XML packages.

An important static analysis problem of a query language is that of optimization, which can reduce to the problem of query containment and query equivalence. In logics closed under boolean

⁸Indeed, this is basically a consequence of considering an XML document as a data tree where the attributes are at leaf positions.

operators, these problems reduce to *satisfiability* checking: does a given query express some property? That is, is there a document where this query has a non-empty result? By answering this question we can decide at compile time whether the query contains a contradiction and thus the computation of the query on the document can be avoided, or if one query can be safely replaced by another one. Moreover, this problem becomes crucial for many applications on security, type checking transformations, and consistency of XML specifications.

Core-XPath (introduced by Gottlob et al. (2005)) is the fragment of XPath that captures all the navigational behavior of XPath. It has been well studied and its satisfiability problem is known to be decidable even in the presence of DTDs. The extension of this language with the possibility to make equality and inequality tests between attributes of elements in the XML document is named **Core-Data-XPath** in (Bojańczyk et al., 2009). The satisfiability problem for this logic is undecidable, as shown by Geerts and Fan (2005). It is then reasonable to study the interaction between different navigational fragments of XPath with equality tests to be able to find decidable and computationally well-behaved fragments.

Definition

XPath is a two-sorted language, with *path* expressions (that we write α, β, γ) and *node* expressions (φ, ψ, η). The fragment $\text{XPath}(\mathcal{O}, =)$, with

$$\mathcal{O} \subseteq \{\downarrow, \downarrow_*, \downarrow_+, \uparrow, \uparrow^*, \uparrow^+, \rightarrow, \rightarrow^*, \rightarrow^+, \leftarrow, \leftarrow^*, \leftarrow^+\}$$

is defined by mutual recursion as follows:

$$\begin{aligned} \alpha, \beta &::= o \mid \alpha[\varphi] \mid [\varphi]\alpha \mid \alpha\beta \mid \alpha \cup \beta & o \in \mathcal{O} \cup \{\varepsilon\}, \\ \varphi, \psi &::= a \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \langle \alpha \rangle \mid \langle \alpha = \beta \rangle \mid \langle \alpha \neq \beta \rangle & a \in \mathbb{A}. \end{aligned}$$

A *formula* of $\text{XPath}(\mathcal{O}, =)$ is either a node expression or a path expression of the logic. $\text{XPath}(\mathcal{O})$ is the fragment of $\text{XPath}(\mathcal{O}, =)$ without the node expressions of the form $\langle \alpha = \beta \rangle$ or $\langle \alpha \neq \beta \rangle$. By $\text{regXPath}(\mathcal{O}, =)$ we refer to the extension of path expressions with the Kleene star operator.⁹

We formally define the semantics of XPath in Table 2. As an example, if \mathbf{t} is the data tree defined by Figure 3, then $\llbracket \langle \downarrow_*[b \wedge \langle \downarrow b \rangle \neq \langle \downarrow b \rangle] \rangle \rrbracket^{\mathbf{t}} = \{\varepsilon, 1, 12\}$, where the formula reads: “*there is a descendant node labeled b , with two children labeled b with different data values.*” We write $\mathbf{t} \models \varphi$ to denote $\llbracket \varphi \rrbracket^{\mathbf{t}} \neq \emptyset$. In this case we say that \mathbf{t} ‘satisfies’ φ . We say that two formulas φ, ψ of XPath are **equivalent** iff $\llbracket \varphi \rrbracket^{\mathbf{t}} = \llbracket \psi \rrbracket^{\mathbf{t}}$ for all data tree \mathbf{t} .

Fragments We define several natural fragments of XPath. Each fragment is defined by the set of axes that the path expressions can use. We call the **downward fragment** of XPath to $\text{XPath}(\downarrow_*, \downarrow, =)$. In fact, all our results apply also to $\text{regXPath}(\downarrow_*, \downarrow, =)$. The **forward fragment** is an extension of the downward fragments with horizontal navigation in only one sense. In our notation, forward XPath is then $\text{XPath}(\downarrow_*, \downarrow, \rightarrow^*, \rightarrow, =)$. The **vertical fragment** is another extension of the downward fragment with upward axes, that is $\text{XPath}(\downarrow_*, \downarrow, \uparrow^*, \uparrow, =)$. Finally, we also show some lower bounds for some **horizontal fragments** of XPath, that is, fragments of XPath running on data words. These lower bounds will then be transferred to fragments that contains horizontal or upwards axes, or that can force a model to be linear (like, *e.g.*, $\text{XPath}(\downarrow_*, \downarrow, \rightarrow, =)$).

⁹ More precisely, by $\text{regXPath}(\mathcal{O}, =)$ we refer to the language where path expressions are extended

$$\alpha, \beta ::= o \mid \alpha[\varphi] \mid [\varphi]\alpha \mid \alpha\beta \mid \alpha \cup \beta \mid \alpha^* \quad o \in \mathcal{O}$$

by allowing the Kleene star on *any* path expression. Although this extension is not enough to already have the expressiveness of MSO—as shown by ten Cate and Segoufin (2008)—, it does give an intuitive language with some counting ability.

$\llbracket \rightarrow \rrbracket^{\mathbf{t}} = \{(x \cdot i, x \cdot (i + 1)) \mid x \cdot (i + 1) \in \text{pos}(\mathbf{t})\}$ $\llbracket \downarrow \rrbracket^{\mathbf{t}} = \{(x, x \cdot i) \mid x \cdot i \in \text{pos}(\mathbf{t})\}$ $\llbracket \alpha^+ \rrbracket^{\mathbf{t}} = \text{the transitive closure of } \llbracket \alpha \rrbracket^{\mathbf{t}}$ $\llbracket \varepsilon \rrbracket^{\mathbf{t}} = \{(x, x) \mid x \in \text{pos}(\mathbf{t})\}$ $\llbracket \alpha \cup \beta \rrbracket^{\mathbf{t}} = \llbracket \alpha \rrbracket^{\mathbf{t}} \cup \llbracket \beta \rrbracket^{\mathbf{t}}$ $\llbracket \alpha[\varphi] \rrbracket^{\mathbf{t}} = \{(x, y) \in \llbracket \alpha \rrbracket^{\mathbf{t}} \mid y \in \llbracket \varphi \rrbracket^{\mathbf{t}}\}$ $\llbracket a \rrbracket^{\mathbf{t}} = \{x \in \text{pos}(\mathbf{t}) \mid \mathbf{a}(x) = a\}$ $\llbracket \neg \varphi \rrbracket^{\mathbf{t}} = \text{pos}(\mathbf{t}) \setminus \llbracket \varphi \rrbracket^{\mathbf{t}}$ $\llbracket \langle \alpha = \beta \rangle \rrbracket^{\mathbf{t}} = \{x \in \text{pos}(\mathbf{t}) \mid \exists y, z (x, y) \in \llbracket \alpha \rrbracket^{\mathbf{t}}, \\ (x, z) \in \llbracket \beta \rrbracket^{\mathbf{t}}, \mathbf{d}(y) = \mathbf{d}(z)\}$	$\llbracket \leftarrow \rrbracket^{\mathbf{t}} = \{(x \cdot (i + 1), x \cdot i) \mid x \cdot (i + 1) \in \text{pos}(\mathbf{t})\}$ $\llbracket \uparrow \rrbracket^{\mathbf{t}} = \{(x \cdot i, x) \mid x \cdot i \in \text{pos}(\mathbf{t})\}$ $\llbracket \alpha^* \rrbracket^{\mathbf{t}} = \text{the reflexive transitive closure of } \llbracket \alpha \rrbracket^{\mathbf{t}}$ $\llbracket \alpha \beta \rrbracket^{\mathbf{t}} = \{(x, z) \mid \text{there exists } y \text{ such that} \\ (x, y) \in \llbracket \alpha \rrbracket^{\mathbf{t}}, (y, z) \in \llbracket \beta \rrbracket^{\mathbf{t}}\}$ $\llbracket [\varphi] \alpha \rrbracket^{\mathbf{t}} = \{(x, y) \in \llbracket \alpha \rrbracket^{\mathbf{t}} \mid x \in \llbracket \varphi \rrbracket^{\mathbf{t}}\}$ $\llbracket \langle \alpha \rangle \rrbracket^{\mathbf{t}} = \{x \in \text{pos}(\mathbf{t}) \mid \exists y. (x, y) \in \llbracket \alpha \rrbracket^{\mathbf{t}}\}$ $\llbracket \varphi \wedge \psi \rrbracket^{\mathbf{t}} = \llbracket \varphi \rrbracket^{\mathbf{t}} \cap \llbracket \psi \rrbracket^{\mathbf{t}}$ $\llbracket \langle \alpha \neq \beta \rangle \rrbracket^{\mathbf{t}} = \{x \in \text{pos}(\mathbf{t}) \mid \exists y, z (x, y) \in \llbracket \alpha \rrbracket^{\mathbf{t}}, \\ (x, z) \in \llbracket \beta \rrbracket^{\mathbf{t}}, \mathbf{d}(y) \neq \mathbf{d}(z)\}$
---	--

Table 2: Semantics of XPath for a data tree $\mathbf{t} = \mathbf{a} \otimes \mathbf{d}$.

Decision problems We study the satisfiability problem for various fragments. That is, for any fixed fragment \mathcal{P} of XPath, the problem of, given a formula $\varphi \in \mathcal{P}$, whether there exists a data tree \mathbf{t} such that $\mathbf{t} \models \varphi$. All the logics we deal with are closed under boolean operations. Hence, the inclusion problem (*i.e.*, the problem of whether $\llbracket \varphi \rrbracket^{\mathbf{t}} \subseteq \llbracket \psi \rrbracket^{\mathbf{t}}$ for every \mathbf{t}) reduces to that of satisfiability. The inclusion problem for φ, ψ yields a ‘yes’ iff the satisfiability problem for $\varphi \wedge \neg \psi$ yields ‘no’. Similarly, the equivalence problem (*i.e.*, the problem of whether $\llbracket \varphi \rrbracket^{\mathbf{t}} = \llbracket \psi \rrbracket^{\mathbf{t}}$ for every \mathbf{t}) reduces to that of satisfiability. Hence, since all the logics we deal with are closed under boolean operations, we only focus in the *satisfiability problem*.

3.2.2 Downward navigation

We investigate the satisfiability problem for downward-XPath, the fragment of XPath that includes the child and descendant axes, and tests for (in)equality of data values. We prove that this problem is decidable, EXPTIME-complete. These bounds also hold when path expressions allow closure under the Kleene star operator. To obtain these results, we introduce a Downward Data automata model (DD automata) over trees with data, which has a decidable emptiness problem. All these results are included in Chapter 5 of the thesis as well as in (Figueira, 2012b, 2009).

Automata model We introduce an automata model that can make rich tests between distant positions of the subtree, but can only perform some limited tests over the order of the siblings. This model has a 2EXPTIME emptiness problem, or a ‘modest’ EXPTIME complexity for some restricted subclass. We call this model the Downward Data automata (or DD automata for short).

A run of a *DD automaton* consists of two steps: (1) the run of a transducer, and (2) the verification of data properties of the transduced tree.

For a data tree $\mathbf{t} = \mathbf{a} \otimes \mathbf{d}$, the first step consists in the translation of \mathbf{a} into another tree \mathbf{b} with the same shape (*i.e.*, the same set of positions). This is done using a nondeterministic letter-to-letter transducer over unranked trees. We adopt a more detailed definition, where the transducer explicitly has as a parameter the class \mathcal{C} of regular properties that it can test over a siblinghood¹⁰ at each transition. If we take this parameter to be the set of all regular properties, this automaton is a standard transducer over unranked trees. However, the emptiness problem for DD automata has a non-primitive-recursive lower bound unless we restrict \mathcal{C} to be a suitable subclass of regular languages. For this reason we define the class of *extensible* languages, that we note \mathcal{E} . One can think of them as all the regular languages defined by regular expressions such that every letter appears under at least one Kleene star (\mathcal{E} -languages are a bit more general).

¹⁰In the context of a tree \mathbf{t} , a *siblinghood* is a maximal sequence of siblings. That is, a sequence of positions $x \cdot 1, \dots, x \cdot l$ of \mathbf{t} such that $x \cdot (l + 1)$ is not a position of \mathbf{t} . That is, it is the sequence of children of a node.

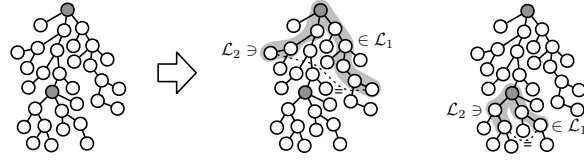


Figure 4: A property tested by the verifier, where the marked nodes are those labeled by a .

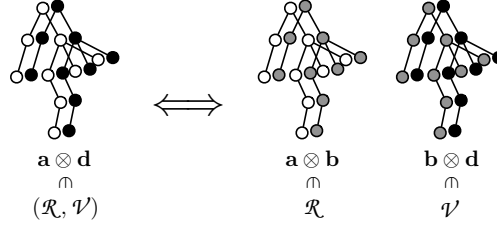


Figure 5: Acceptance condition of a DD automaton $(\mathcal{R}, \mathcal{V})$.

In the second step, for every subtree of the transduced tree $\mathbf{b} \otimes \mathbf{d}$, a property on the data values of the tree is verified. We call this automaton the *verifier*. The letter at the root of the subtree under inspection determines the property to verify. The properties are boolean combinations of tests verifying the existence of data values shared by nodes in the subtree, hanging from branches satisfying some regular expression. This automaton can test, for example, that all the nodes labeled with an ‘ a ’ have two descendants with equal data value, one reachable with a path whose labeling is in a regular language \mathcal{L}_1 , and another in a language \mathcal{L}_2 , as in Figure 4.

A \mathcal{C} -Downward Data automaton (\mathcal{C} -DD for short) is then a pair $(\mathcal{R}, \mathcal{V})$ made of a \mathcal{C} -transducer $\mathcal{R} \subseteq \text{Trees}(\mathbb{A} \times \mathbb{B})$ and a verifier $\mathcal{V} \subseteq \text{Trees}(\mathbb{B} \times \mathbb{D})$. A data tree $\mathbf{a} \otimes \mathbf{d}$ is accepted by $(\mathcal{R}, \mathcal{V})$ iff there exists $\mathbf{b} \in \text{Trees}(\mathbb{B})$ such that $\mathbf{a} \otimes \mathbf{b} \in \mathcal{R}$ and $\mathbf{b} \otimes \mathbf{d} \in \mathcal{V}$, as depicted in Figure 5. The main result is that if we restrict \mathcal{C} to be the class of extensible languages \mathcal{E} , then the emptiness problem is decidable in 2EXPTIME .

Theorem 5.9. The emptiness problem for \mathcal{E} -DD automata is in 2EXPTIME .

The proof of this theorem is divided into four parts. In the first part (§5.3.1) we define some decoration or marking of the nodes of a data trees that in some sense witnesses the acceptance of a run of a DD automaton. These decorations of the tree are the main structures with which we work with in our proof.

The second part (§5.3.2) is dedicated to proving two properties. The first property states that if a DD automaton is nonempty, it accepts a tree decorated with some guidance system that marks the paths to be covered in order to verify the properties imposed by the verifier. In some sense, it decorates the tree as in Figure 4, avoiding having two paths going through the same node. The guidance system is called *certificate* and this property is called *admissibility of correct certificates*. The second property states that if a DD automaton is nonempty, then it accepts a tree whose data values are in a certain normal form, as follows: Every pair of subtrees rooted at two different children of a node, have a disjoint set of data values, with the exception of some polynomially bounded many (this is called the *disjoint values property*).

The third part (§5.3.3) is centered around proving that DD automata have the exponential width model property. That is, if a DD automaton has a nonempty language, then it accepts a tree whose width is exponentially bounded in the size of the automaton.

In the fourth part (§5.3.4) we give the algorithm for testing emptiness of DD automata, which is based on the bound on the width and the two other properties: the disjoint values property and

\downarrow	\downarrow_*	$=$	Complexity	Details
•			PSPACE-complete	Cor. 5.50
	•		PSPACE-complete	Thm. 5.54
•	•		EXPTIME-complete	Marx (2004)
•		•	PSPACE-complete	Prop. 5.49
	•	•	EXPTIME-complete	Thm. 5.45, Thm. 5.46
•	•	•	EXPTIME-complete	Thm. 5.45, Thm. 5.46
regXPath($\downarrow, =$)			EXPTIME-complete	Thm. 5.45, Thm. 5.46
XPath ^ℓ ($\downarrow_*, =$)			PSPACE-complete	Prop. 5.56
regXPath($\downarrow, =$) + \mathcal{E}_{tree}			EXPTIME-complete	Thm. 5.57, Thm. 5.46

Table 3: Summary of results on downward XPath.

the admissibility of correct certificates.

Downward XPath We show that any XPath($\downarrow_*, \downarrow, =$) formula —moreover, any regXPath($\downarrow, =$) formula— can be efficiently translated into an equivalent \mathcal{E} -DD automaton. Although this automata model has a 2EXPTIME emptiness problem, it can be shown to be decidable in EXPTIME when restricted to the sub-class of automata needed to capture regXPath($\downarrow, =$). In this way we obtain an EXPTIME procedure of the satisfiability for regXPath($\downarrow, =$).

In fact, \mathcal{E} -DD automata are more expressive than regXPath($\downarrow, =$), and this is true even for the aforementioned sub-class of automata. For example, although regXPath($\downarrow, =$) does not include any horizontal axis, \mathcal{E} -DD automata can test for certain horizontal properties. Whereas typical properties that can be expressed by both downward XPath and \mathcal{E} -DD are (T1), (T6), and (T7), the \mathcal{E} -DD model can further express, for instance, that the sequence of children of the root is described by the regular expression $(abc)^*$. It then follows that the satisfiability problem for regXPath($\downarrow, =$) under the regular constraints that can be expressed by \mathcal{E} -DD automata remains decidable in EXPTIME. This is a particularly well-behaved class of regular properties, since the satisfiability problem of regXPath($\downarrow, =$) restricted to regular tree languages is known to have non-primitive-recursive complexity.¹¹

Finally, we give the exact complexity of the satisfiability problem for several fragments of downward-XPath. We prove that the fragment XPath($\downarrow_*, =$) without the \downarrow axis is EXPTIME-hard, even for a restricted fragment of XPath($\downarrow_*, =$) without unions of path expressions. This reduction seems to rely on data equality tests, as the corresponding fragment XPath(\downarrow_*) without unions is shown to be PSPACE-complete. We thus prove that the satisfiability problems for XPath($\downarrow_*, =$), XPath($\downarrow_*, \downarrow, =$) and regXPath($\downarrow, =$) are all EXPTIME-complete. Additionally, we present a natural fragment of XPath($\downarrow_*, =$) that is PSPACE-complete (named XPath^ℓ($\downarrow_*, =$)). We complete the picture by showing that satisfiability for XPath($\downarrow, =$) is also PSPACE-complete. Our results, together with the results of Benedikt et al. (2008) and Marx (2004), establish the precise complexity for all downward fragments of XPath with and without data tests (see Table 3).

3.2.3 Forward navigation

We investigate logics and automata for data trees with a *forward* behavior, in the sense that we can not only move downwards in the tree, but we can also navigate (in only one sense) the sequence of siblings. We extend the model ARA(guess, spread) to a model ATRA(guess, spread) of Alternating Tree Register Automata that run over data trees (instead of data words). The decidability of the emptiness follows from an extension of the well-quasi-ordering argument for the decidability result for ARA(guess, spread). As in the case of data trees, this model allows to show the decidability of a logic.

¹¹This is a corollary of our lower bounds on data words.

From the logics perspective, we investigate the satisfiability of forward XPath. The satisfiability problem for this logic follows by a reduction to the emptiness problem of $\text{ATRA}(\text{guess}, \text{spread})$. This reduction is not trivial, since XPath is closed under negation and our automata model is not closed under complementation. Indeed, $\text{ATRA}(\text{guess}, \text{spread})$ and forward XPath have incomparable expressive power. These results are included in Chapter 6 of the thesis, as well as in (Figueira, 2010, 2012a).

Automata model An Alternating Tree Register Automaton (ATRA) consists in a top-down tree walking automaton with alternating control and *one* register to store and test data. This model is essentially the same automaton presented for data words, that works on a (unranked, ordered) *data tree* instead of a data word. The only difference is that instead of having one instruction \triangleright that means “move to the next position”, we have two instructions \triangleright and ∇ meaning “move to the next sibling to the right” and “move to the leftmost child”. Its emptiness problem is known to be decidable (Jurdziński and Lazić, 2008), as in the case of ARA, through a reduction to a class of incrementing counter automata on data trees. We simplify the proof of (Jurdziński and Lazić, 2008) as we did for the ARA model, easily obtaining decidability by a minor extension to the proof of emptiness for ARA. Here, as in the case of data words, we consider an extension with the operators *spread* and *guess*. We call this model $\text{ATRA}(\text{spread}, \text{guess})$.

As for ARA, the ATRA model is closed under all boolean operations (Jurdziński and Lazić, 2008). However, the extensions introduced *guess* and *spread*, while adding expressive power, are not closed under complementation as a trade-off for decidability.

We show that the emptiness problem for this model is decidable, extending the approach used for ARA and show the decidability of the two extensions *spread* and *guess*. This model of computation enables us to show decidability of a large fragment of XPath.

Forward XPath We prove the decidability of the satisfiability problem for the forward fragment of XPath, that is $\text{XPath}(\downarrow, \downarrow_*, \rightarrow, \rightarrow^*, =)$.

Jurdziński and Lazić show that ATRA captures a fragment of forward XPath where for all subformulas $\langle \alpha = \beta \rangle$ and $\langle \alpha \neq \beta \rangle$ we have $\alpha = \epsilon$. We call this fragment $\text{XPath}^\epsilon(\downarrow, \downarrow_*, \rightarrow, \rightarrow^*, =)$.¹² ATRA can easily capture the Kleene star operator on any path formula, obtaining decidability of $\text{regXPath}^\epsilon(\downarrow, \downarrow_*, \rightarrow, \rightarrow^*, =)$. However, these decidability results cannot be generalized to the full unrestricted forward fragment $\text{XPath}(\downarrow, \downarrow_*, \rightarrow, \rightarrow^*, =)$ as ATRA is not powerful enough to capture the full expressivity of the logic. Neither ATRA nor $\text{regXPath}^\epsilon(\downarrow, \downarrow_*, \rightarrow, \rightarrow^*, =)$ can express, for instance, that there are two different leaves with the same data value. On the other hand, $\text{ATRA}(\text{guess}, \text{spread})$ can express this property. But it cannot express the *negation* of the property! It is worth noting that $\text{XPath}(\downarrow, \downarrow_*, \rightarrow, \rightarrow^*, =)$, contrary to $\text{XPath}^\epsilon(\downarrow, \downarrow_*, \rightarrow, \rightarrow^*, =)$, can express unary *key constraints* (i.e., whether for some symbol a , all the a -elements in the tree have different data values) like (T5), as well as (T6), (T7).

Indeed, the $\text{ATRA}(\text{guess}, \text{spread})$ model cannot capture $\text{XPath}(\downarrow, \downarrow_*, \rightarrow, \rightarrow^*, =)$. Indeed, data tests of the form $\neg \langle \alpha = \beta \rangle$ are impossible to perform for $\text{ATRA}(\text{guess}, \text{spread})$ as this would require—in some sense—the ability to guess two disjoint sets of data values S_1, S_2 such that all α -paths lead to a data value of S_1 , and all β -paths lead to a data value of S_2 . Still, we show that there exists a reduction from the satisfiability of $\text{regXPath}(\downarrow, \downarrow_*, \rightarrow, \rightarrow^*, =)$ to the emptiness of $\text{ATRA}(\text{guess}, \text{spread})$, and hence that the former problem is decidable. This result settles an open question left in (Jurdziński and Lazić, 2008) regarding the decidability of the satisfiability problem for forward XPath. The main result is the following.

Theorem 6.9. Satisfiability of $\text{regXPath}(\downarrow, \downarrow_*, \rightarrow, \rightarrow^*, =)$ in the presence of DTDs (or any regular language) and unary key constraints is decidable, non-primitive-recursive.

¹²By $\text{XPath}^\epsilon(\mathcal{O}, =)$ (resp. $\text{regXPath}^\epsilon(\mathcal{O}, =)$) we denote the fragment of $\text{XPath}(\mathcal{O}, =)$ (resp. of $\text{regXPath}(\mathcal{O}, =)$) where node expressions are defined by $\varphi, \psi ::= a \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \langle \alpha \rangle \mid \langle \epsilon = \alpha \rangle \mid \langle \epsilon \neq \alpha \rangle$ for $a \in \mathbb{A}$. That is, the data tests are performed between the data value of the *current* node and some node accessed by α .

To show this, we prove that for every formula φ of $\text{regXPath}(\downarrow, \downarrow_*, \rightarrow, \rightarrow^*, =)$ there is a computable $\text{ATRA}(\text{guess}, \text{spread})$ automaton \mathcal{A} such that \mathcal{A} is nonempty iff φ is satisfiable. This proof can be sketched as follows:

1. We show that for every nonempty automaton $\mathcal{A} \in \text{ATRA}(\text{guess}, \text{spread})$ there is an accepting run on a data tree with the *disjoint values property*, which is a property of the data values of the tree with respect to the data values appearing in the run.
2. We give an effective translation from a given forward XPath formula η to an $\text{ATRA}(\text{guess}, \text{spread})$ automaton \mathcal{A} such that: (1) any tree accepted by a run of the automaton \mathcal{A} with the disjoint values property verifies the XPath formula η , and (2) any tree verified by the formula η is accepted by a run of the automaton \mathcal{A} with the disjoint values property.

3.2.4 Vertical navigation

Two-way automata on data words and trees have frequently an undecidable emptiness problem. We introduce a decidable automata model that, while being bottom-up, presents several features that allows to make tests by navigating the tree in both directions: upwards and downwards. This two-way flavor is witnessed by the fact that these automata can decide vertical XPath.

We introduce a novel decidable class of automata over unranked data tree, that we denote BUDA, for Bottom-Up alternating 1-register Data tree Automata. The BUDA are essentially alternating bottom-up tree automata with one register, without the ability of testing for “horizontal” properties on the siblings of the tree, such as for example bounding the rank of the tree. However, an automaton of this class has the ability to test rich data properties on the subtrees, which in some sense corresponds to a downward behavior.

The decidability of this automaton is proven using a WSTS, with somewhat similar techniques as for ARA and ATRA models. However, finding the correct wqo that is compatible with the automaton is not easily derivable from the automaton’s run. Since the automaton can faithfully simulate an ARA when going up to the root, the complexity of the emptiness problem is necessarily non-primitive-recursive. We stress that the absence of horizontal tests is essential to obtain our decidability results. In fact, one can see that the model would become undecidable could it force a bound on the tree’s rank.

As a result of the “two-wayness” flavor of the automata model, it can capture the vertical fragment of XPath. The vertical fragment $\text{XPath}(\downarrow, \downarrow_*, \uparrow, \uparrow^*, =)$ is the one containing downward axes \downarrow, \downarrow_* and upward axes \uparrow, \uparrow^* . Our main result on XPath is then the following.

Theorem 7.1. The satisfiability problem for $\text{regXPath}(\downarrow, \downarrow_*, \uparrow, \uparrow^*, =)$ is decidable.

In this way we answer positively to the open question raised by Benedikt and Koch (2008, Question 5.10) and Benedikt et al. (2008), regarding the decidability of the satisfiability for vertical XPath.

All these results are joint work with Luc Segoufin and are contained in Chapter 7 of the thesis as well as in (Figueira and Segoufin, 2011).

Automata model We consider a *bottom-up* alternating tree automata with one register. Although the automaton is one-way, it has features that allow to test data properties that can navigate the tree in both directions: upward and downward. We call this automaton model BUDA.

The BUDA model is essentially a bottom-up tree automata with one register and an alternating control and 1 register. We show that these automata are at least as expressive as vertical XPath.

We aim at defining a decidable class of automata that can express data properties both in the downwards and the upwards directions. To obtain such a model of automata, the switch from top-down to bottom-up is essential. As a result, this class can capture vertical XPath, and in particular is expressively incomparable with respect to ATRA or $\text{ATRA}(\text{guess}, \text{spread})$. (It also makes the decidability of its emptiness problem significantly more difficult.) The ATRA and

ATRA(guess, spread) automata models are top-down instead of bottom-up, and they can test for horizontal properties. For example, they can express that every node has at most one child, something that cannot be tested by BUDA. On the other hand, BUDA can test properties like (T7), that cannot be expressed by these models of automata, or any *inclusion dependency constraint* property such as (T4). But, on the other hand, the BUDA automata cannot test for a property on the siblings.

An automaton $\mathcal{A} \in \text{BUDA}$ that runs over data trees of $\text{Trees}(\mathbb{A} \times \mathbb{D})$ is defined as a tuple $\mathcal{A} = (\mathbb{A}, \mathbb{B}, Q, q_0, \delta_\epsilon, \delta_{up}, \mathcal{S}, h)$ where \mathbb{A} is the finite alphabet of the tree, \mathbb{B} is an internal finite alphabet of the automaton (whose purpose will be clear later), Q is a finite set of states, q_0 is the initial state, \mathcal{S} is a finite semigroup, h is a semigroup homomorphism from $(\mathbb{A} \times \mathbb{B})^+$ to \mathcal{S} , δ_ϵ is the ϵ -transition function of \mathcal{A} , and δ_{up} is the up -transition function of \mathcal{A} .

δ_{up} is a partial function from states to formulas. For $q \in Q$, $\delta_{up}(q)$ is either undefined or a formula consisting in a disjunction of conjunctions of states. δ_ϵ is also a partial function from states to disjunctions of conjunctions of ‘atoms’ of one of the following forms:

$$p \mid \text{guess}(p) \mid \text{univ}(p) \mid \text{store}(p) \mid \text{eq} \mid \overline{\text{eq}} \mid \\ \mid \langle \mu \rangle^= \mid \langle \mu \rangle^\neq \mid \overline{\langle \mu \rangle^=} \mid \overline{\langle \mu \rangle^\neq} \mid \text{root} \mid \overline{\text{root}} \mid \text{leaf} \mid \overline{\text{leaf}} \mid a \mid \bar{a} \mid b \mid \bar{b}$$

where $\mu \in \mathcal{S}$, $p \in Q$, $a \in \mathbb{A}$, $b \in \mathbb{B}$.

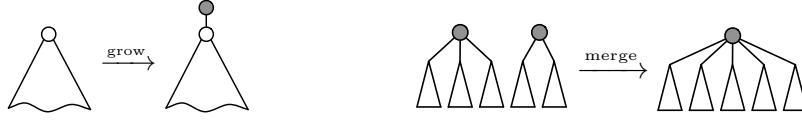
We first describe the battery of tests the automata can perform. All these tests are explicitly closed under negation, denoted with the $\overline{}$ notation, and of course they are also closed under intersection and union using the alternating and nondeterministic control of the automata. The automata can test the label and internal label of the current node and also whether the current node is the root, a leaf or an internal node. The automata can test (in)equality of the current data value with the one stored in the register (eq and $\overline{\text{eq}}$). Finally the automata can test the existence of some downward path, starting from the current node and leading to a node whose data value is (or is not) equal to the one currently stored in the register, such that the path satisfies some regular property on the labels. These properties are specified using the finite semigroup \mathcal{S} and the morphism $h : (\mathbb{A} \times \mathbb{B})^+ \rightarrow \mathcal{S}$ over the words made of the label of the tree and the internal label. For example, $\langle \mu \rangle^=$ tests for the existence of a path that evaluates to μ via h , which starts at the current node and leads to a node whose data value matches the one currently stored in the register. Similarly, $\langle \mu \rangle^\neq$ tests that it leads to a data value different from the one currently in the register.

Based on the result of these tests, the automata can perform the following actions. They can change state, store the current data value in the register ($\text{store}(p)$), or store an arbitrary data value nondeterministically chosen ($\text{guess}(p)$). Finally, a transition can demand to start a new thread in state p for every data value of the subtree with the operation $\text{univ}(p)$. The automata can also decide to move up in the tree according to the up -transition function.

We stress that the automata model is not closed under complementation because its set of actions are not closed under complementation: guess is a form of existential quantification while univ is a form of universal quantification, but they are not dual. Actually, adding any of their dual would yield undecidability of the model. We show that this model is decidable.

Theorem 7.5. The emptiness problem for BUDA is decidable.

In order to achieve this, we associate to each BUDA a WSTS that simulates its runs. The transition system works on sets of *abstract configurations*. Given an automaton, an abstract configuration is meant to contain all the information that is necessary to collect at the root of a given subtree in order to continue the simulation of the automaton from there. The aforesaid transition system works with *sets* of such abstract configurations in order to capture the bottom-up behavior of the automaton on unranked trees. The transition relation of the WSTS essentially corresponds to the transitions of the automaton except for the up -transition. An up -transition of

Figure 6: The *grow* and *merge* operations.

the automaton is simulated by a succession of two types of transitions of the WSTS, called *grow* and *merge*. The object of doing this is to avoid having transitions that take an unbounded number of arguments (as the *up* relation in the run of the automaton does). The *grow* transition adds a node on top of the current root, and the *merge* transition identifies the roots of two abstract configurations. Intuitively, these transitions correspond to the operations on trees of Figure 6. This is necessary because we do not know in advance the arity of the tree and therefore the transition system has to build one subtree at a time. We then exhibit a well-quasi-order (wqo) on abstract configurations and show that the transition system is compatible with respect to this wqo.

Vertical XPath We prove that the class BUDA captures $\text{regXPath}(\downarrow, \downarrow_*, \uparrow, \uparrow^*, =)$, and hence that vertical-XPath has a decidable satisfiability problem. Given a formula η of $\text{regXPath}(\downarrow, \downarrow_*, \uparrow, \uparrow^*, =)$, we say that a BUDA \mathcal{A} is *equivalent* to η if a data tree \mathbf{t} is accepted by \mathcal{A} iff $\llbracket \eta \rrbracket^{\mathbf{t}} \neq \emptyset$. We then obtain the following.

Proposition 7.25. For every $\eta \in \text{regXPath}(\downarrow, \downarrow_*, \uparrow, \uparrow^*, =)$ there exists an equivalent $\mathcal{A} \in \text{BUDA}$ computable from η .

The idea is that it is easy to simulate any positive test $\langle \alpha = \beta \rangle$ or $\langle \alpha \neq \beta \rangle$ of vertical XPath by a BUDA using $\langle \mu \rangle^=$ and $\langle \mu \rangle^{\neq}$. For example, consider the property $\langle \downarrow_*[a] \neq \uparrow \downarrow[b] \rangle$, which states that there is a descendant labeled a with a different data value than a sibling labeled b . A BUDA automaton can test this property as follows: (1) It guesses a data value d and stores it in the register. (2) It tests that d can be reached by $\downarrow_*[a]$ with a test $\langle \mu \rangle^=$ for a suitable μ . (3) It moves up to its parent. (4) It tests that a different value than d can be reached in one of its children labeled with b , using the test $\langle \mu \rangle^{\neq}$ for a suitable μ .

The simulation of negative tests ($\neg \langle \alpha = \beta \rangle$ or $\neg \langle \alpha \neq \beta \rangle$) is more tedious as BUDA is not closed under complementation. Nevertheless, the automaton has enough universal quantifications (in the operations univ , $\overline{\langle \mu \rangle^=}$ and $\overline{\langle \mu \rangle^{\neq}}$) in order to do the job, through a more involved coding.

We mention that even though $\text{XPath}(\downarrow, \downarrow_*, \rightarrow, \rightarrow^*, =)$ (*i.e.*, forward-XPath) on data words has a non-primitive-recursive complexity, the results of (Figueira, 2011) suggest that $\text{XPath}(\downarrow, \downarrow_*, \rightarrow^*, =)$ or even $\text{XPath}(\downarrow, \downarrow_*, \rightarrow^*, * \leftarrow, =)$ may be decidable in elementary time (*cf.* Figueira, 2011, Conjecture 1).

3.2.5 XPath on data words

In our work on lower bounds for LTL^\downarrow (in §3.6 of the thesis and (Figueira and Segoufin, 2011)), we identify a fragment of $\text{LTL}^\downarrow(\mathbf{F})$, called *simple*, that has a very simple navigation, and that has connections with the logic XPath. In fact, they have the same expressive power. Then, our lower bounds for LTL^\downarrow yield the following bounds (contained in §4.5 of the thesis).

Logic	Complexity	Details
$\text{XPath}(\downarrow_+, \rightarrow, =)$	non-primitive-recursive	Corollary 4.3
$\text{XPath}(\downarrow_+, \uparrow^+, =)$	non-primitive-recursive	Corollary 4.5
$\text{XPath}(\downarrow_+, \uparrow^+, \rightarrow, =)$	undecidable	Corollary 4.7
$\text{XPath}(\rightarrow^+, \downarrow, \uparrow, =)$	undecidable	Corollary 4.8

The strictness of the axes —that is, the fact of having \uparrow^+ instead of \uparrow^* — seems to be necessary. Surprisingly, whereas $\text{XPath}(\rightarrow^+, =)$ has a non-primitive-recursive lower bound, $\text{XPath}(\rightarrow^*, =)$ has

a 2EXPSpace upper bound (Figueira, 2011). Moreover, whereas $\text{XPath}(\rightarrow^+, * \leftarrow, =)$ is undecidable, $\text{XPath}(\rightarrow^*, * \leftarrow, =)$ is decidable in 2EXPSpace (Figueira, 2011)!¹³

4 Conclusions

This work had as objective the development of techniques and decidable formalisms to work with data values. We have seen several automata models that are decidable over data trees, and one over data words. We introduced formalisms that can express different kind of properties, and are on the limit of decidability.

Automata All the automata for data trees introduced are incomparable in expressive power (*cf.* Chapter 8 of the thesis), and they exploit the tree structure of the model. For example, the BUDA automata would have an undecidable emptiness problem would it run on data words, or even k -ranked trees for some k , and similarly the DD automata model would have a non-primitive-recursive emptiness problem. We also mention that for some automata (ARA, ATRA, BUDA) we showed connections with well-structured transition systems, devising new techniques to interpret runs of automata as a transition system with certain compatibility properties.

Logics On XPath, we showed decidability of the downward, forward, and vertical fragments, thus settling some open questions. We have now a clearer landscape of the decidability status of XPath according to the set of axes that it uses. The main results are summarized in Table 4. In the presence of DTDs (or regular languages) we obtain that the forward and downward XPath fragments are decidable with a non-primitive recursive lower bound, and that vertical XPath is undecidable. These results are stated in terms of XPath fragments but they must also be seen more generally as results about logics that navigate trees with data. For example, our results on downward-XPath could also be applied to a logic like CTL with one register to store and compare data values (like in LTL^\downarrow) and some restricted policy of testing for data values.

We remark that although we showed that each of the aforesaid fragments is decidable, this does not mean that we can *combine* these results. Our results do not yield the possibility to test the satisfiability of a boolean combination of formulas where each of them belong to one of these fragments. Indeed, this problem is undecidable.

Perspectives

All our investigation on XPath focuses on the satisfiability problem. As already mentioned, the problem of query equivalence and inclusion reduce to this problem. But this concerns only queries of *node* expressions. Whether the techniques we developed can be adapted to show similar decidability results on the problems of query equivalence and query containment of *path* expressions is a moot point. We leave then the question: What is the decidability status of the inclusion and equivalence problems of path expressions of downward, forward and vertical XPath?

The fragments treated here are all navigational fragments of XPath 1.0. However XPath 2.0 has many rich features that we do not consider. We leave open the question of whether the results of this thesis can be extended to incorporate some of the distinctive features of XPath 2.0.

Another relevant issue is to try to add more domain specific relations to our models of automata. In that direction, we discussed that a linear order can be added to $\text{ARA}(\text{guess}, \text{spread})$ without losing decidability. Moreover, with the same kind of analysis it can be further extended to $\text{ATRA}(\text{guess}, \text{spread})$. It would be interesting to explore other relations. For example if the data domain is the set of strings $\mathbb{D} = \mathbb{A}^*$, we may want to have the substring, prefix, and suffix relations; and if it is numerical $\mathbb{D} = \mathbb{N}$ we may want to use some arithmetic. A possible future work would be hence to extend the automata treated here with some relations or functions of this kind.

¹³The 2EXPSpace upper bound result (Figueira, 2011) is not part of the thesis.

↓	↓+	↑	↑+	→	→+	←	←+	Complexity
•								PSPACE-complete
	•							EXPTIME-complete
•	•							EXPTIME-complete
•	•			•	•			Decidable, NPR
•	•	•	•					Decidable, NPR
			•					Decidable, NPR
	•		•					Decidable, NPR
	•		•	•				Undecidable
					•		•	Undecidable
•		•			•			Undecidable

Table 4: Summary of main results on XPath with data values. NPR stands for a non-primitive recursive lower bound.

References

- Michael Benedikt, Wenfei Fan, and Floris Geerts. XPath satisfiability in the presence of DTDs. *Journal of the ACM (JACM)*, 55(2):1–79, 2008. doi:10.1145/1346330.1346333.
- Michael Benedikt and Christoph Koch. XPath leashed. *ACM Computing Surveys*, 41(1), 2008. doi:10.1145/1456650.1456653.
- Henrik Björklund and Mikołaj Bojańczyk. Bounded depth data trees. In *International Colloquium on Automata, Languages and Programming (ICALP’07)*, volume 4596 of *Lecture Notes in Computer Science*, pages 862–874. Springer, 2007. doi:10.1007/978-3-540-73420-8_74.
- Henrik Björklund and Thomas Schwentick. On notions of regularity for data languages. *Theoretical Computer Science*, 411(4-5):702–715, 2010. doi:10.1016/j.tcs.2009.10.009.
- Mikołaj Bojańczyk and Sławomir Lasota. An extension of data automata that captures XPath. In *Annual IEEE Symposium on Logic in Computer Science (LICS ’10)*, 2010.
- Mikołaj Bojańczyk, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data trees and XML reasoning. *Journal of the ACM (JACM)*, 56(3):1–48, 2009. doi:10.1145/1516512.1516515.
- Mikołaj Bojańczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin, and Claire David. Two-variable logic on words with data. In *Annual IEEE Symposium on Logic in Computer Science (LICS’06)*, pages 7–16. IEEE Computer Society Press, 2006. doi:10.1109/LICS.2006.51.
- Patricia Bouyer, Antoine Petit, and Denis Thérien. An algebraic approach to data languages and timed languages. *Inf. Comput.*, 182(2):137–162, 2003. doi:10.1016/S0890-5401(03)00038-5.
- Balder ten Cate and Luc Segoufin. XPath, transitive closure logic, and nested tree walking automata. In *ACM Symposium on Principles of Database Systems (PODS’08)*, pages 251–260. ACM Press, 2008. doi:10.1145/1376916.1376952.
- James Clark and Steve DeRose. XML path language (XPath). Website, 1999. W3C Recommendation. <http://www.w3.org/TR/xpath>.
- Stéphane Demri and Ranko Lazić. LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic (TOCL)*, 10(3), 2009. doi:10.1145/1507244.1507246.
- Stéphane Demri, Ranko Lazić, and David Nowak. On the freeze quantifier in constraint LTL: Decidability and complexity. In *International Symposium on Temporal Representation and Reasoning (TIME’05)*, pages 113–121. IEEE Computer Society Press, 2005. doi:10.1016/j.ic.2006.08.003.
- Diego Figueira. Satisfiability of downward XPath with data equality tests. In *ACM Symposium on Principles of Database Systems (PODS’09)*, pages 197–206. ACM Press, 2009. doi:10.1145/1559795.1559827.
- Diego Figueira. Forward-XPath and extended register automata on data-trees. In *International Conference on Database Theory (ICDT’10)*. ACM Press, 2010. doi:10.1145/1804669.1804699.

- Diego Figueira. A decidable two-way logic on data words. In *Annual IEEE Symposium on Logic in Computer Science (LICS'11)*, pages 365–374, Toronto, Canada, June 2011. IEEE Computer Society Press. doi:10.1109/LICS.2011.18.
- Diego Figueira. Alternating register automata on finite data words and trees. *Logical Methods in Computer Science (LMCS)*, 8(1:22), 2012a. doi:10.2168/LMCS-8(1:22)2012.
- Diego Figueira. Decidability of downward XPath. *ACM Transactions on Computational Logic (TOCL)*, 13(4), 2012b. To appear.
- Diego Figueira, Santiago Figueira, Sylvain Schmitz, and Philippe Schnoebelen. Ackermannian and primitive-recursive bounds with Dickson’s lemma. In *Annual IEEE Symposium on Logic in Computer Science (LICS'11)*, pages 269–278, Toronto, Canada, June 2011. IEEE Computer Society Press. doi:10.1109/LICS.2011.39.
- Diego Figueira, Piotr Hofman, and Slawomir Lasota. Relating timed and register automata. In *International Workshop on Expressiveness in Concurrency (EXPRESS'10)*, 2010. doi:10.4204/EPTCS.41.5.
- Diego Figueira and Luc Segoufin. Future-looking logics on data words and trees. In *International Symposium on Mathematical Foundations of Computer Science (MFCS'09)*, volume 5734 of *LNCS*, pages 331–343. Springer, 2009. doi:10.1007/978-3-642-03816-7_29.
- Diego Figueira and Luc Segoufin. Bottom-up automata on data trees and vertical XPath. In *International Symposium on Theoretical Aspects of Computer Science (STACS'11)*. Springer, 2011.
- Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001. doi:10.1016/S0304-3975(00)00102-X.
- Floris Geerts and Wenfei Fan. Satisfiability of XPath queries with sibling axes. In *International Symposium on Database Programming Languages (DBPL'05)*, volume 3774 of *Lecture Notes in Computer Science*, pages 122–137. Springer, 2005. doi:10.1007/11601524_8.
- Georg Gottlob, Christoph Koch, and Reinhard Pichler. Efficient algorithms for processing XPath queries. *ACM Transactions on Database Systems*, 30(2):444–491, 2005. doi:10.1145/1071610.1071614.
- Marcin Jurdziński and Ranko Lazić. Alternating automata on data trees and XPath satisfiability. *Computing Research Repository (CoRR)*, 2008. arXiv:0805.0330.
- Michael Kaminski and Nissim Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2): 329–363, 1994. doi:10.1016/0304-3975(94)90242-9.
- Michael Kaminski and Tony Tan. Tree automata over infinite alphabets. In *Pillars of Computer Science*, volume 4800 of *Lecture Notes in Computer Science*, pages 386–423. Springer, 2008. doi:10.1007/978-3-540-78127-1_21.
- M.H. Löb and S.S. Wainer. Hierarchies of number theoretic functions, I. *Archiv für Mathematische Logik und Grundlagenforschung*, 13:39–51, 1970. doi:10.1007/BF01967649.
- Maarten Marx. XPath with conditional axis relations. In *International Conference on Extending Database Technology (EDBT'04)*, volume 2992 of *Lecture Notes in Computer Science*, pages 477–494. Springer, 2004. doi:10.1007/b95855.
- Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Transactions on Computational Logic (TOCL)*, 5(3):403–435, 2004. doi:10.1145/1013560.1013562.
- Thomas Schwentick and Thomas Zeume. Two-variable logic with two order relations. In *EACSL Annual Conference on Computer Science Logic (CSL'10)*, 2010.