Complexity of Data Tree Patterns over XML Documents

Claire David

LIAFA, University Paris 7 and CNRS, France
cdavid@liafa.jussieu.fr

Abstract. We consider Boolean combinations of data tree patterns as a specification and query language for XML documents. Data tree patterns are tree patterns plus variable (in)equalities which express joins between attribute values. Data tree patterns are a simple and natural formalism for expressing properties of XML documents. We consider first the model checking problem (query evaluation), we show that it is DP-complete[1] in general and already NP-complete when we consider a single pattern. We then consider the satisfiability problem in the presence of a DTD. We show that it is in general undecidable and we identify several decidable fragments.

1 Introduction

The relational model and its popular query language SQL are widely used in database systems. However, it does not fit well in the ever changing Internet environment, since its structure is fixed by an initially specified schema which is difficult to modify. When exchanging and manipulating large amounts of data from different sources, a less structured and more flexible data model is preferable. This was the initial motivation for the Extensible Markup Language (XML) model which is now the standard for data exchange.

An XML document is structured as an unranked, labelled tree. The main difference with the relational model is that in XML, data is also extracted because of its position in the tree and not only because of its value. Consequently, all the tools manipulating XML data, like XML query languages and XML schema, combine navigational features with classical data extraction ones. XPath[2] is a typical example. It has a navigational core, known as Core-XPath and studied in [16], which is essentially a modal language that walks around in the tree. XPath also allows restricted tests on data attributes. It is the building block of most XML query languages (XQuery, XSLT...). Similarly, in order to specify integrity constraints in XML Schema, XML languages have navigational features for description of walks in the tree and selection of nodes. The nodes are for instance chosen according to a key or a foreign key [15].

[1] A problem DP is the intersection of a NP problem and a co-NP problem.
[2] In all the paper, XPath refers to XPath1.0

In this paper, we study an alternative formalism as a building block for querying and specifying XML data. It is based on Boolean combinations of data tree patterns. A data tree pattern is essentially a tree with child or descendant edges, labelled nodes and (in)equality constraints on data values. Intuitively, a document satisfies a data tree pattern if there exists an injective mapping from the tree pattern into the tree that respects edges, node labels and data value constraints. Using patterns, one can express properties on trees in a natural, visual and intuitive way. These properties can express queries, as well as some integrity constraints.

At first glance, the injectivity requirement does not seem important; however, it has some consequences in terms of expressive power. As we do not consider horizontal order between siblings, without injectivity data tree patterns are invariant by bisimulation. Data tree patterns with injective semantics are strictly more expressive than with non-injective semantics. For example, it is not possible to express desirable properties such as a node has two a-labelled children without injectivity. Another consequence of injectivity appears when considering conjunctions of data tree patterns. With non-injective semantics, the conjunction of two patterns would be equivalent to a new pattern obtained by merging the two patterns at the root. With injectivity this no longer works and we have to consider conjunctions of tree patterns. This difference appears when we study the complexity of the satisfiability problem: for one pattern the problem is PTime while it is untractable for a conjunction of patterns.

XPath and data tree patterns are incomparable in terms of expressiveness. Without data value, XPath queries are closed under bisimulation while data tree patterns are not. On the other hand, XPath allows negation of subformulas while we only allow negation of a full data tree pattern. For example XPath can check whether a node has a-labelled children but no b-labelled child. This is not possible with Boolean combinations of tree pattern. In terms of data comparison, Xpath allows very limited joins because XPath queries cannot compare more than two elements at a time, while a single pattern can compare simultaneously an arbitrary number of elements.

In this paper, to continue this comparison, we study the complexity of two questions related to data tree patterns: the model checking problem (query evaluation) and the satisfiability problem in the presence of schema.

The evaluation of XPath queries has been extensively studied (see [6] for a detailed survey). The evaluation problem is PTime for general XPath queries. In our case, this problem is more difficult: the combined complexity of the model checking problem for Boolean combinations of data tree patterns is untractable. We prove that it is DP-complete in general and already NP-complete when considering only one tree pattern.

The satisfiability problem for XPath is undecidable in general [5]. However for many fragments the problem is decidable with a complexity ranging from NP to NExpTime. Similarly, for Boolean combinations of data tree patterns the satisfiability problem is undecidable in general. We identify several decidable fragments by restraining the expressivity of tree patterns or by bounding

the depth of the documents. The corresponding complexities range from NP to 2ExpTime.

Related Work: Tree patterns have already been investigated in a database context, often without data values [22, 3, 20]. The focus is usually optimisation techniques for efficient navigation [1, 12, 7]. In this work, we focus on the difficulty raised by data values and we are not interested in optimisation but in the worst case complexity for the model checking and satisfiability problems.

Several papers considered the non injective semantics of tree pattern with data constraints. First, [19] considered the satisfiability problem for one positive pattern while we consider Boolean combinations of tree pattern. Then, the authors of [2] consider the type checking problem which is more powerful that unsatisfiability but incomparable to the satisfiability problem.

Data tree patterns are used in [4] to specify data exchange settings. They study two problems: the first one is consistency of data exchange settings, the second one is query answering under data exchange settings. Given a conjunction of data tree patterns and a DTD, we can construct a data exchange setting such that the consistency of this setting is equivalent to the satisfiability of the conjunction of patterns in the presence of the DTD. However the data tree patterns they consider are less expressive than ours, in that they can not express inequality constraints on data values nor Boolean combinations of data tree patterns. The other problem considered in this paper is query answering. This problem seems related to our model checking problem. However it does not seem possible to use their result or their proof techniques.

Fragment of XPath: In [14], the authors consider an XPath fragment (simple XPath) allowing only vertical navigation but augmented with data comparisons. Negation is disallowed, both in the navigation part and in the comparison part. A simple XPath expression can be viewed as a pattern with non-injective semantics and only data equality. They study the inclusion problem of such expressions wrt special schemes (SXIC) containing integrity constraints like inclusion dependency. We cannot simulate inclusion dependency even with Boolean combinations of data tree patterns. Hence, their framework is incomparable to ours.

Conjunctive queries on trees: Conjunctive queries on trees can be expressed by tree patterns. They were considered in [17, 8] without data values. Very recently [9], an extension by schema constraints is proposed and in very few cases they allow data comparison. Notice that, without sibling predicate, those conjunctive queries are strictly less expressive than our framework because they do not allow negation and do not have an injective semantic. It is shown that the query satisfiability problem is NP-complete, whereas the query validity problem is 2ExpTime-complete. Moreover, the validity of a disjunction of conjunctive queries is shown to be undecidable. This last result corresponds to our undecidability result but the proof is different.

Logics over infinite alphabets: Another related approach is to consider logic for trees over an infinite alphabet. In [10, 11], the authors study an extension of First Order Logic with two variables. In [13, 18], the focus is on temporal logic and μ-calculus. These works are very elegant, but the corresponding complexities

are non primitive recursive. Our work can be seen as a continuation of this work aiming for lower complexities.

Structure: Section 2 contains the necessary definitions. In Section 3, we consider the model-checking problem. In Section 4, we consider the satisfiability problem in general and the restricted cases. Section 5 contains a summary of our results and a discussion. Omitted proofs can be found in the appendix available at `http://www.liafa.jussieu.fr/~cdavid/publi/mfcs08.pdf`.

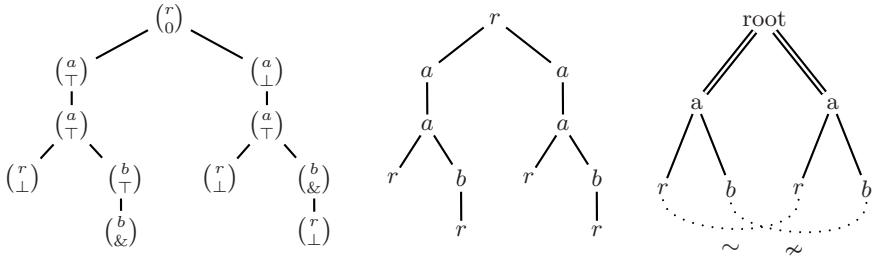
2 Preliminary

In this paper, we consider XML documents that are modeled as unordered, unranked *data trees*, as considered e.g. in [10].

Definition 1 A **data tree** over a finite alphabet Σ is an unranked, unordered, labelled tree with data values. Every node v has a *label* $v.l \in \Sigma$ and a *data value* $v.d \in \mathcal{D}$, where $\mathcal{D}$ is an infinite domain.

We only consider equality tests between data values. The data part of a tree can thus be seen as an equivalence relation $\sim$ on its nodes. In the following, we write $u \sim v$ for two nodes u, v, if $u.d = v.d$ and we use the term *class* without more precision to denote an equivalence class for the relation $\sim$.

The **data erasure** of a data tree t over Σ is the tree obtained from t by ignoring the data value $v.d$ of each node v of t.



(a) data tree (b) data erasure (c) pattern

Fig. 1. Examples

Data tree patterns are a natural way to express properties of data trees, or to query such trees. They describe a set of nodes through their relative positions in the tree, and (in)equalities between their data values.

Definition 2 A **data tree pattern** $P = (p, C_\sim, C_{\not\sim})$ consists of:

- an unordered, unranked tree p, with nodes labelled either by Σ or by the wildcard symbol $*$, and edges labelled either by | (child edges) or by ‖ (descendant edge), and
- two binary relations $C_\sim$ and $C_{\not\sim}$ on the set of nodes of p.

A data tree t satisfies a pattern $P = (p, C_\sim, C_{\not\sim})$, and we write $t \models P$, if there exists an *injective* mapping f from the nodes of p to the nodes of t that is consistent with the labelling, the relative positions of nodes, the branching structure and the data constraints. Formally, we require the following:

- for every node v from p with $v.l \in \Sigma$, we have $v.l = f(v).l$,
- for every pair of nodes (u,v) from p, if $(u, v) \in C_\sim$ (resp. $(u, v) \in C_{\not\sim}$) then $f(u) \sim f(v)$ (resp. $f(u) \not\sim f(v)$),
- for every pair of nodes (u,v) from p, if (u, v) is an edge of p labelled by $|$ (resp. by $\|$), then $f(v)$ is a child (resp. a descendant) of $f(u)$,
- for any nodes u,v,z from p, if (u, v) and (u, z) are both edges of p labelled by $\|$, then $f(v)$ and $f(z)$ are *not* related by the descendant relation in t.

A mapping f as above is called a **witness** of the pattern P in the data tree t.

Notice that the semantic does not preserve the least common ancestor and asks for an injective mapping between the nodes of a pattern and those of the tree. This enables patterns to express integrity constraints. We will discuss the impact of those choices in Section 5. Data tree patterns can describe properties that XPath cannot, see *e.g.* the pattern in Fig 1 (XPath cannot talk simultaneously about the two r-nodes and the two b-nodes).

We denote by $Ptn(\sim, |, \|)$ the set of data tree patterns and by $\mathrm{BC}(\sim, |, \|)$ the set of Boolean combinations of data tree patterns. We will also consider restricted patterns, that do not use child relations or do not use descendant relations (denoted respectively by $Ptn(\sim, \|)$, $Ptn(\sim, |)$). From these, we derive the corresponding classes of Boolean combinations. Finally, BC^+ (resp. BC^-) denotes conjunctions of patterns (resp. negations of patterns).

In proofs, we consider the *parse tree* of a Boolean formula φ over patterns, denoted by $\mathcal{T}(\varphi)$. The leaves of this tree are labelled by (possibly negation of) patterns and inner nodes are labelled by conjunctions or disjunctions. Such trees are of linear size in the size of the formula and can be computed in PTime.

Given a pattern formula from $\mathrm{BC}(\sim, |, \|)$, the main problems we are interested in are the model-checking on a data tree (evaluation), the satisfiability problem, in the general case as well as for interesting fragments. Because the general structure of XML documents is usually constrained, we may consider DTDs as additional inputs. DTDs are essentially regular constraints on the finite structure of the tree. Since we work on unordered, unranked trees, we use as DTDs an unordered version of hedge automata. A DTD is a bottom-up automaton $\mathcal{A}$ where the transition to a state q' with label a is given by a Boolean combination of clauses of the form $\#q \leq k$ where q is a state and k a constant (unary encoded). A clause $\#q \leq k$ is satisfied if there are at most k children in state q. Adding a DTD constraint does not change the complexity results for the model-checking, since checking whether the data erasure of a tree satisfies a DTD is PTime. Therefore, we do not mention DTDs in the model-checking part. We consider the following problems:

Problem 1. Given a data tree t and a pattern formula φ, the **model-checking** problem asks whether t satisfies φ.

Problem 2. Given a pattern formula φ and a DTD L, the **satisfiability** problem in the presence of a DTD asks whether φ is satisfied by some data tree whose data erasure belongs to L.

3 Model Checking

Patterns provide a formalism for expressing properties. In this section, we see how efficiently we can evaluate them. Our main result is the exact complexity of the model-checking problem for pattern formulas from BC($\sim, |, \|$).

Theorem 3 *The model-checking problem for BC($\sim, |, \|$) is DP-complete.*

The class of complexity DP is defined as the class of problems that are the conjunction of a NP problem and a co-NP problem [21]. In particular, DP includes both NP and co-NP. A typical DP-complete problem is SAT/UNSAT: given two propositional formulas φ_1, φ_2, it asks whether φ_1 is satisfiable, and φ_2 is unsatisfiable.

The key to the proof of Theorem 3 is the case where only one pattern is present. This problem is already NP-complete.

Proposition 4 The model-checking problem for a single pattern from $Ptn(\sim, |, \|)$ is NP-complete.

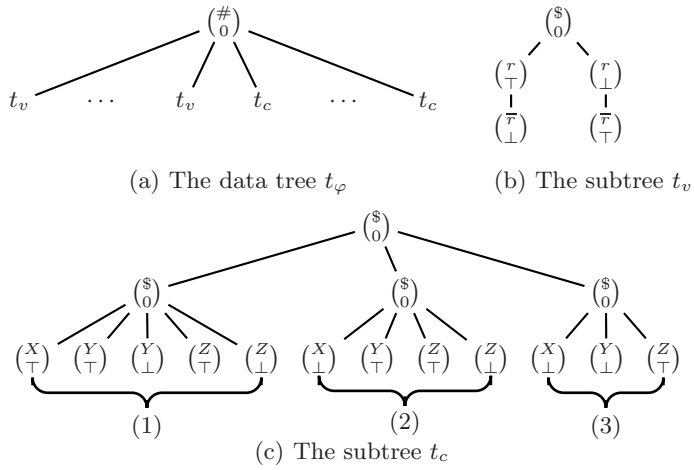
Proof. The upper bound is obtained by an algorithm guessing a witness for the pattern in the data tree and checking in PTime whether the witness is correct. The lower bound is more difficult. It is obtained by a reduction of 3SAT.

Given a propositional formula φ in 3-CNF, we build a data tree t_φ and a pattern P_φ of polynomial size, such that $t_\varphi \vDash P_\varphi$ iff φ is satisfiable. Because we consider the model-checking problem, the data tree is fixed in the input. Thus, it must contain all possible valuations of the variables and at least all possible true valuations of each variable. Moreover, one positive data tree pattern should identify a true valuation of the formula and check its consistency. Hence, it does not seem possible to use previously published encodings of 3SAT into trees.

The pattern selects one valuation per variable and per clause. Its structure ensures that only one valuation per variable and per literal is selected. The constraints on data ensure the consistency of the selection. The data tree and the tree of the pattern depend only on the number of variables and clauses of the formula. Only the constraints on data of the pattern are specific to the formula. They encode the link between variables and clauses.

Let $\Sigma = \{r, \overline{r}, X, Y, Z, \#, \$\}$ be the finite alphabet. Assume that φ has k variables and n clauses. The data tree t_φ is composed of k copies of the tree t_v and n copies of the tree t_c as depicted in Figure 2. Even if we consider unordered trees, each copy of t_v corresponds to a variable of the formula and each copy of t_c to a clause. The tree t_φ involves exactly three classes, denoted as $0, \top, \bot$.

Each subtree t_v, see Figure 2(b), contains the two possible values for a variable. The left (right) branch of the tree represents true (resp. false).



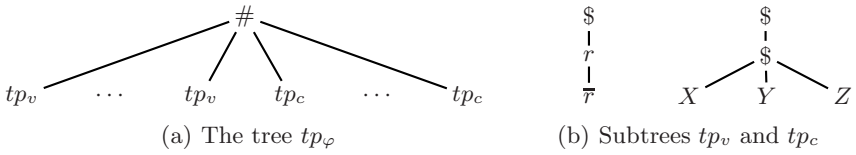
(a) The data tree t_φ (b) The subtree t_v

(c) The subtree t_c

Fig. 2. The data tree t_φ

A clause is viewed here as the disjunction of three literals, say X, Y, and Z. Each subtree t_c, see Figure 2(c), is formed by three subtrees. Each of them represents one of the three disjoint possibilities for a clause to be true: (1) X is true, or (2) X is false and Y is true, or (3) X and Y are false and Z is true.

We now turn to the definition of the tree pattern $P_\varphi = (tp_\varphi, C_\sim, C_{\not\sim})$, depicted in Figure 3. Similarly to t_φ, the tree tp_φ is formed by k copies of tp_v (each of them implicitly corresponding to a variable) and n copies of tp_c (each of them implicitly corresponding to a clause).



(a) The tree tp_φ (b) Subtrees tp_v and tp_c

Fig. 3. The tree tp_φ

The form of the data erasures of t_φ and tp_φ ensures that any witness of P_φ in t_φ selects exactly one value per variable and one (satisfying) valuation for each clause. Note that this is ensured by the definition of witness, since the witness mapping is injective.

It remains to define the data constraints $C_\sim$ and $C_{\not\sim}$ in order to guarantee that each clause is satisfied. Assume that the first literal of clause c is a positive variable x (resp. the negation of x). Then we add in $C_\sim$ the r-position (resp. the $\overline{r}$-position) of the subtree tp_v corresponding to the variable x together with the X-position of the subtree tp_c corresponding to the clause c. The same can be

done with the literals Y and Z. Figure 4 gives the example of the pattern for the formula φ with only the clause $a \vee \neg b \vee c$.

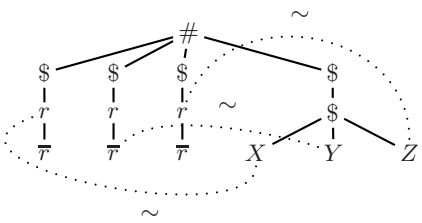


Fig. 4. Pattern for $\varphi = a \vee \neg b \vee c$

We now prove that $t_\varphi \vDash P_\varphi$ iff φ is satisfiable. Assume that the formula φ is satisfiable. From any satisfying assignment of φ we derive a mapping of P_φ into t_φ: the subtree p_v corresponding to the variable v is mapped on the left branch of the corresponding t_v if the value of v is true, and on the right branch otherwise. Since each clause is satisfied, one of the three cases represented by the subtree t_c happens, and we can map the tp_c corresponding to the clause on the branch of the corresponding t_c. The converse is similar. □

In the proofs of Proposition 4 and Theorem 3, the patterns use only the child predicate. We can do the same with similar patterns using only the descendant predicate. As a consequence, we have:

Theorem 5 *The model-checking problem for both fragments BC*$(\sim, \|)$*, BC*$(\sim, |)$ *is DP-complete.*

Corollary 6 The model-checking problem for $\mathrm{BC}^+(\sim, \|)$, $\mathrm{BC}^+(\sim, |)$ and for $\mathrm{BC}^+(\sim, |, \|)$ is NP-complete.

Similarly, we can see that the model-checking problem of a (conjunction of) negated pattern(s) is co-NP-complete. Notice that in the proof of Theorem 3, the pattern formula of the lower bound is a conjunction of one pattern and the negation of one pattern. Thus, the model-checking problem is already DP-complete for a conjunction of one pattern and one negated pattern.

The model checking problem for conjunctive queries is also exponential (NP) in relational databases. However, the algorithms work very well in practice, when models or queries are simple. In particular, when the query is acyclic, the problem becomes polynomial. The worst cases that lead to exponential behaviors do not appear often. It would be interesting to know how the algorithms following from our proofs behave on practical cases, and whether we can find some restriction on the patterns that would lead to efficient evaluation in practice.

4 Satisfiability

In this section, we study the satifiability problem in the presence of DTDs. Checking satisfiability of a query is useful for optimization of query evaluation

or minimization techniques. In terms of schema design, satisfiability corresponds to checking the consistency of the specification.

We show that the satisfiability problem is undecidable in general. However the reduction needs the combination of negation, child and descendant operations. Indeed, removing any one of these features yields decidability, and we give the corresponding precise complexities.

4.1 Undecidability

Theorem 7 *The satisfiability problem for BC*$(\sim, |, \|)$ *in the presence of DTD is undecidable.*

Proof sketch. We prove the undecidability by a reduction from the acceptance problem of two-counter machines (or Minsky machines). Our reduction builds a DTD and a pattern formula of size polynomial in the size of the machine whose models are exactly the encodings of the accepting runs.

The encoding of a run can be split in three parts:

1. The general structure of the tree, which depends only on the data erasure, and is controlled by the DTD.
2. The internal consistency of a configuration.
3. The evolution of counter values between two successive configurations.

The global structure contains a branch that is labelled by the sequence of transitions. Ensuring that a tree is of this shape is done by the DTD. It recognizes the data erasure of sequences of configurations. In particular it checks that a counter is zero when this is required by the transition. It also ensures that the sequence of transitions respects the machine's rules (succession of control states, initial and final configurations).

The data values allow us to control the evolution of the counters between two consecutive configurations. In order to do so, we need to guarantee a certain degree of structure and continuity of the values through a run. The data structure and the evolution of counters are ensured by the pattern formula. □

The proof uses only conjunctions of negated patterns. Thus, the satisfiability problem is already undecidable for the $\mathrm{BC}^{-}(\sim, |, \|)$ fragment in the presence of a DTD. Alternatively, the DTD can be replaced by a pattern formula. To do so, we need a few positive patterns to constrain initial and final configurations in the coding. Thus, the satisfiability problem is undecidable for $\mathrm{BC}(\sim, |, \|)$ without DTDs. It is interesting to notice that the satisfiability problem of $\mathrm{BC}(\sim, |, \|)$ is undecidable on word models. We will discuss this in Section 5.

4.2 Decidable Restrictions

We can obtain decidability by restraining either the expressive power of pattern formulas or the data trees considered. For the first part, using only one kind of edge predicate ($|$ or $\|$) leads to decidability. For the second part, restricting the trees to bounded depth leads to decidability. We provide the exact complexities.

Restricted Fragments: The proof of undecidability uses both $\|$ and $|$ in the pattern to count unbounded values of the counters. If we restrict expressivity of patterns to use either $\|$ or $|$, we can't do this anymore and the problem becomes decidable. The key to both lower bounds is that patterns can still count up to a polynomial value and thus compare positions of a tree of polynomial depth. We use this idea to encode exponential size configurations of a Turing machine into the leaves of polynomial depth subtrees.

Theorem 8 *The satisfiability problem of BC*$(\sim, |)$ *in the presence of a DTD is* 2ExpTime*-complete.*

Proof sketch. The upper bound is obtained by a small model property. We can prove that a pattern formula φ of BC$(\sim, |)$ is satisfiable in the presence of a given DTD iff it has a model with a number of classes that is doubly exponential in the size of the formula. We can recognize the data erasure of such small models with an automaton of size doubly exponential in the size of the formula. Because emptiness of such automata is PTime, we have the 2ExpTime upper bound.

The lower bound is obtained by a coding of accepting runs of AExpSpace Turing machines. We can build a DTD and a pattern formula from BC$(\sim, |)$ such that a data tree is a model on the pattern formula and respects the DTD iff it is the encoding of an accepting run of the machine. □

Theorem 9 *The satisfiability problem of BC*$(\sim, \|)$ *in the presence of a DTD is* NExpTime*-complete.*

Bounded Depth restrictions: In the context of XML documents, looking at the satisfiability problem restricted to data trees of bounded depth is a crucial restriction. This restriction leads to decidability for BC$(\sim, <, +1)$.

Problem 3. Consider a pattern formula φ, an integer d and a DTD L. The problem of **bounded depth satisfiability in the presence of a DTD** asks whether φ is satisfiable by a data tree of depth smaller than d whose data erasure belongs to L.

Theorem 10 *If d is fixed, the bounded depth satisfiability problem in the presence of a DTD for BC*$(\sim, \|, |)$ *is Σ_2-complete.*

Theorem 11 *If d is part of the input, the bounded depth satisfiability problem in the presence of a DTD for BC*$(\sim, \|, |)$ *is* NExpTime*-complete.*

Other remarks: All the lower bound results of this section only use conjunctions of negated patterns. Thus these results hold for the BC$^-$ fragments.

Proposition 12 The satisfiability problem of a single pattern is PTime.

Proposition 13 The satisfiability problem for BC$^+(\sim, |, \|)$ is NP-complete in the presence of DTD.

5 Conclusion

The table below summarizes our results. *bnd* (resp. bnd_f) Sat stands for Bounded depth Satisfiability when the bound is part of the input (resp. fixed). The gray parts of the table gives complexity results for data words models. Data words are the linear model corresponding to data trees. This model is studied in the verification area [11, 13]. Data patterns can also be considered for data words. The proofs are more complex and will be available in a longer version.

Fragments	Model-Checking	Satisfiability	*bnd* Sat	bnd_f Sat
BC($\sim, \|\|, \|$)	DP-complete	Undecidable	NExpTime-complete	Σ_2-complete
BC($\sim, \|$)	DP-complete	2ExpTime-complete	NExpTime-complete	Σ_2-complete
Data Word	PTime	PSpace-complete		
BC($\sim, \|\|$)	DP-complete	NExpTime-complete	NExpTime-complete	Σ_2-complete
Data Word	DP-complete	Σ_2-complete		
$BC^-(\sim, \|\|, \|)$	coNP-complete	Undecidable	NExpTime-complete	Σ_2-complete
Data Word	coNP-complete	undecidable		
$BC^+(\sim, \|\|, \|)$	NP-complete	NP-complete	NP-complete	NP-complete
Data Word	NP-complete	NP-complete		

Discussion:

- In our framework we use the unordered version of trees. If we consider the next-sibling predicate, the situation is different. For the model checking problem all results hold with similar proofs. However, the complexity of the satisfiability problem can increase when negation is allowed. In particular the satisfiability problem for bounded depth tree becomes undecidable since we can encode data words.
- Recall that our pattern formalism does not preserve the least common ancestor. All results hold if we add the least common ancestor.
- An important issue of semi-structured databases is the containment problem. Given a DTD and two pattern formulas we want to know whether every tree satisfying the DTD and the first formula also satisfies the second one. When the set of formulas we consider is closed under negation, we can decide whether a formula φ_1 is more constraining than φ_2 by checking the satisfiability of $\varphi_2 \wedge \neg\varphi_1$. In Boolean combinations, we have closure under negation, hence the inclusion problem reduces to the satisfiability problem. For the positive fragment, the precise complexity seems harder to state and the question is left open.
- In terms of expressiveness, our pattern formalism is incomparable to XPath. In terms of tractability, evaluation of XPath queries is PTime whereas model-checking of one data tree pattern is already NP-hard. A question is to find good notions of constraints in order to isolate interesting fragments with lower complexity. Considering the complexity of the satisfiability problem, XPath and our pattern formalism behave similarly.
- In this paper, we only consider patterns as filters in order to define properties on the data trees. Defining a query language would be a natural extension of this work. To do this, some of the variables of the patterns can be chosen as output variables.

References

1. S. Al-Khalifa, H. V. Jagadish, J. M. Patel, Y. Wu, N. Koudas, and D. Srivastava. Structural Joins: A Primitive for Efficient XML Query Pattern Matching. In *ICDE*, pages 141–153. IEEE, 2002.
2. N. Alon, T. Milo, F. Neven, D. Suciu, and V. Vianu. XML with data values: typechecking revisited. *J. Comput. Syst. Sci.*, 66(4):688–727, 2003.
3. S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, and D. Srivastava. Minimization of tree pattern queries. *SIGMOD Rec.*, 30(2):497–508, 2001.
4. M. Arenas and L. Libkin. XML data exchange: consistency and query answering. In *PODS*, pages 13–24, 2005.
5. M. Benedikt, W. Fan, and F. Geerts. XPath satisfiability in the presence of DTDs. In *PODS*, pages 25–36, 2005.
6. M. Benedikt and C. Koch. XPath Leashed. To appear in ACM Computing Surveys.
7. V. Benzaken, G. Castagna, and C. Miachon. CQL: a pattern-based query language for XML. In *BDA*, pages 469–490, 2004.
8. H. Björklund, W. Martens, and T. Schwentick. Conjunctive Query Containment over Trees. In *DBPL*, LNCS 4797, pages 66–80. Springer, 2007.
9. H. Björklund, W. Martens, and T. Schwentick. Optimizing Conjunctive Queries over Trees using Schema Information. To appear in MFCS, 2008.
10. M. Bojanczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data trees and XML reasoning. In *PODS*, pages 10–19, 2006.
11. M. Bojanczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David. Two-Variable Logic on Words with Data. In *LICS*, pages 7–16. IEEE, 2006.
12. N. Bruno, N. Koudas, and D. Srivastava. Holistic twig joins: optimal XML pattern matching. In *SIGMOD Conference*, pages 310–321. ACM, 2002.
13. S. Demri and R. Lazic. LTL with the Freeze Quantifier and Register Automata. In *LICS*, pages 17–26. IEEE, 2006.
14. A. Deutsch and V. Tannen. Containment and integrity constraints for xpath. In *KRDB*, volume 45 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2001.
15. W. Fan and L. Libkin. On XML integrity constraints in the presence of DTDs. *J. ACM*, 49(3):368–406, 2002.
16. G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. *ACM Trans. Database Syst.*, 30(2):444–491, 2005.
17. G. Gottlob, C. Koch, and K. U. Schulz. Conjunctive queries over trees. *J. ACM*, 53(2):238–272, 2006.
18. M. Jurdzinski and R. Lazic. Alternation-free modal mu-calculus for data trees. In *LICS*, pages 131–140. IEEE, 2007.
19. L. V. S. Lakshmanan, G. Ramesh, H. Wang, and Z. J. Zhao. On Testing Satisfiability of Tree Pattern Queries. In *VLDB*, pages 120–131, 2004.
20. A. Neumann and H. Seidl. Locating matches of tree patterns in forests. In *FSTTCS*, volume 1530 of *LNCS*, pages 134–145. Springer, 1998.
21. C. H. Papadimitriou and M. Yannakakis. The Complexity of Facets (and Some Facets of Complexity). *J. Comput. Syst. Sci.*, 28(2):244–259, 1984.
22. Y. Wu, J. M. Patel, and H. V. Jagadish. Structural Join Order Selection for XML Query Optimization. In *ICDE*, pages 443–454. IEEE, 2003.

A Model-Checking

Here we prove Theorem 3:

The model-checking problem for BC($\sim, |, \|$) is DP-complete.

Proof. The upper bound is obtained by a straightforward algorithm. Consider a data tree t and a pattern formula φ. From Proposition 4 model-checking a data tree pattern is in NP and model-checking a negated pattern is in co-NP. Thus we have a DP procedure for any formula φ from BC($\sim, |, \|$) to fill the parse tree. Each leaf labeled by a positive pattern is evaluated in NP, and each leaf labeled by a negative pattern is evaluated in co-NP. Checking the parse tree of the pattern formula is PTime.

For the lower bound, we reduce SAT/UNSAT to our problem, using the encoding defined in the proof of Proposition 4. We use two disjoint copies of the alphabet Σ to encode the two formulas. The instance (φ, ψ) of SAT/UNSAT is the data tree t constructed by merging the roots of t_φ and t_ψ. We have $t \models P_\varphi \wedge \neg P_\psi$ iff φ is satisfiable and ψ is not satisfiable. □

B Satisfiability

B.1 Undecidability

Here we show the Theorem 7:

The satisfiability problem for BC($\sim, |, \|$) in presence of DTD is undecidable.

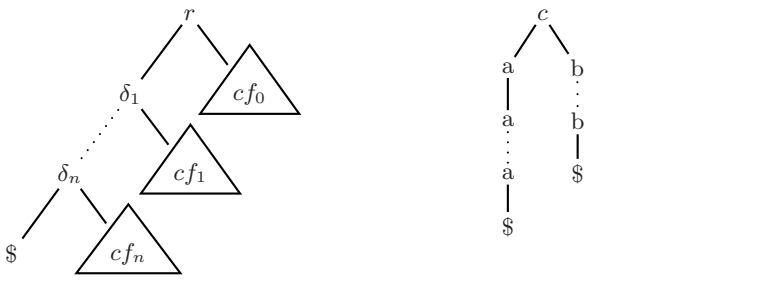
Proof. We prove the undecidability by a reduction from the acceptance problem of two-counter machines (or Minsky machines). Our reduction builds a DTD and a pattern formula such that any data tree satisfying both is the coding of an accepting run of the machine. In the following, we explain how to encode an accepting run into a data tree and we describe the DTD and the pattern formula.

The encoding of a run can be split in three parts:

1. The general structure of the tree, which depends only on the data erasure, and is controlled by the DTD.
2. The internal consistency of a configuration.
3. The evolution of counter values between two successive configurations.

The global structure is described in Figure 5(a). A run contains a branch that is labeled by the sequence of transitions. Each transition branches to a subtree corresponding to the current configuration. The subtree corresponding to a configuration is made of two branches, one with only a's and one final \$ (the a-branch) and the other with only b's and another final \$ (the b-branch), as shown in Figure 5(b). They represent, in unary form, the values of the counters. Ensuring that a tree is of this shape is done by the DTD. It recognizes the data erasure of sequences of configurations. In particular it checks that a counter is

zero when this is required by the transition. It also ensures that the sequence of transitions respects the machine's rules (succession of control states, initial and final configurations).



(a) The encoding of a run (b) The encoding of a configuration cf_i

Fig. 5. Encoding of a run

The data values allow us to control the evolution of the counters between two consecutive configurations. In order to do so, we need to guarantee a certain degree of structure and continuity of the values through a run. We show how this can be done for the a-branches. Similar patterns control the behavior of the b-branches.

The first step is to ensure that in any configuration, the data values in the a-branches are all different. The pattern described in Figure 6(a) corresponds to the case where two a's in the same configuration have the same value, and so we forbid it.

The second step brings in the necessary continuity: if two a's are at the same position in two successive configurations, they must share the same data value. Forbidding the two patterns of Figure 6(b) ensures inductively that a divergence cannot occur.

The last step is to check that the lengths of the branches change according to the current transition. Two patterns, corresponding to the cases where there are too many or too few patterns in the new configuration, are forbidden. The precise form of these patterns depends on whether the counter increases, decreases, or remains constant. Figure 6(c) describes the ones for a transition that increases the first counter.

We have now all the elements of the reduction. We can build a DTD and a pattern formula of size polynomial in the size of the machine whose models are exactly the encodings of the accepting runs. □

B.2 Descendant Fragment

Here we show Theorem 9:

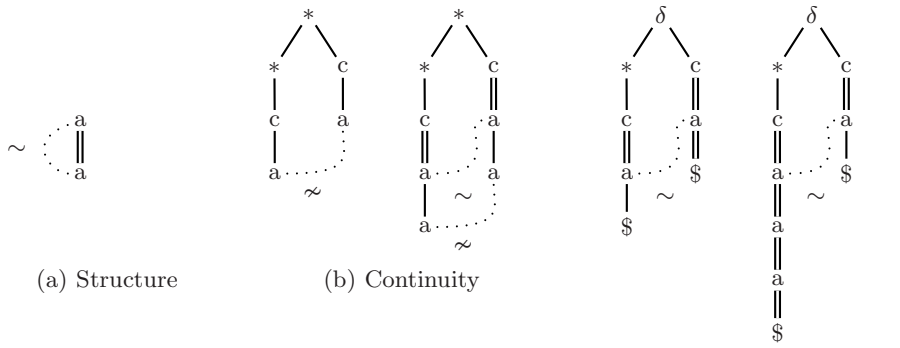


Fig. 6. Forbidden patterns

The satisfiability problem of BC($\sim$, $\|$) in the presence of a DTD is NExpTime-complete.

Proof. The upper bound result follows from a small model property. If a pattern formula is satisfiable in the presence of a given DTD, there exists a model of rank and depth polynomial in the size of the formula and the DTD. Therefore the model has a size exponential in the input.

Consider a pattern formula φ and a DTD L. Consider a model t of φ that respects the DTD L. Mark the nodes of one witness for each pattern of φ satisfied by t. Ignoring data values, as in classical regular tree languages, we can pump w.r.t. the DTD, both horizontally and vertically, any part of the tree that does not contain a marked position. This way, we reduce the rank and depth of the tree to be polynomial into the size of L and the patterns of the formula φ. The data erasure of the new tree still belongs to L. The new tree contains all the previous witnesses. Since the patterns only allow descendant predicates, if the original tree does not contain a pattern, the new tree does not either. Thus, the new tree is still a model of the formula.

A straightforward algorithm answers the satisfiability problem in NExpTime. First guess a data tree t of exponential size. For each pattern of the formula, consider each possible mapping of the pattern into the tree (there are $|t|^k$ possible matchings if k is the size of the pattern) and check whether the matching is a witness for the pattern (PTime in the size of the tree $|t|$). Finally, evaluate the syntactic parsing tree of φ (PTime in the size of the formula).

The lower bound result is obtained by a reduction from the acceptance of a non-deterministic Turing machine that uses time 2^n to our problem. This yields NExpTime-hardness. We encode a run of a NExpTime Turing machine into a data tree. Then we explain how to build a DTD and a pattern formula whose models are exactly the encodings of the accepting runs of the machine.

Consider a machine $\mathcal{M}$ and an input w of length n. The machine $\mathcal{M}$ uses time 2^n. An accepting run of the machine $\mathcal{M}$ on input w is a succession of at most 2^n transitions $\delta_0, \delta_1, \delta_2, \delta_3, \ldots, \delta_f$. Moreover each configuration is of length

at most 2^n. We encode such a run with a tree of the following form given in Figure 7:

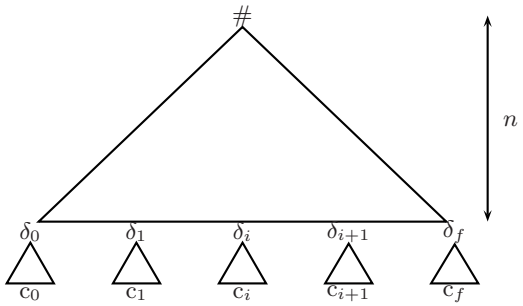


Fig. 7.

The first subtree rooted by # is a binary tree of depth n. This is a tree where each node has exactly one child labeled by 0 and the other one labeled by 1. In each leaf, we plug another binary tree of depth n (rooted by a transition of the machine) such that the leaf word corresponds to a configuration of the machine (consider leaves ordered by the lexical order on the label of the path from the root). In the following, we denote those subtrees by $\mathcal{C}$-subtrees. All $\mathcal{C}$-subtrees have the same label structure and data values. The only difference lies in the labels of leaves allowing to encode configurations. Moreover two nodes belonging to the same $\mathcal{C}$-subtree must have different data values.

The DTD ensures rank equal to two, the shape of the sub-trees, the paths labeled 0 and 1. The pattern formula ensures the data value property and checks transitions between configurations. Because corresponding positions in all $\mathcal{C}$-subtrees have the same sequence of data values along the path from the root, we can identify them in the patterns and then compare the evolution of consecutive positions between consecutive configurations. As in the proof of Theorem 7, we only forbid bad behaviors.

We first explain how to enforce the data value properties described above.

- In a $\mathcal{C}$-subtree each position is in a different class.This is ensured by forbidding the patterns given in Figure 8:
- A position has the same data value in every $\mathcal{C}$-subtree. This is ensured by forbidding the patterns of Figure 9.

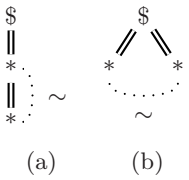


Fig. 8.

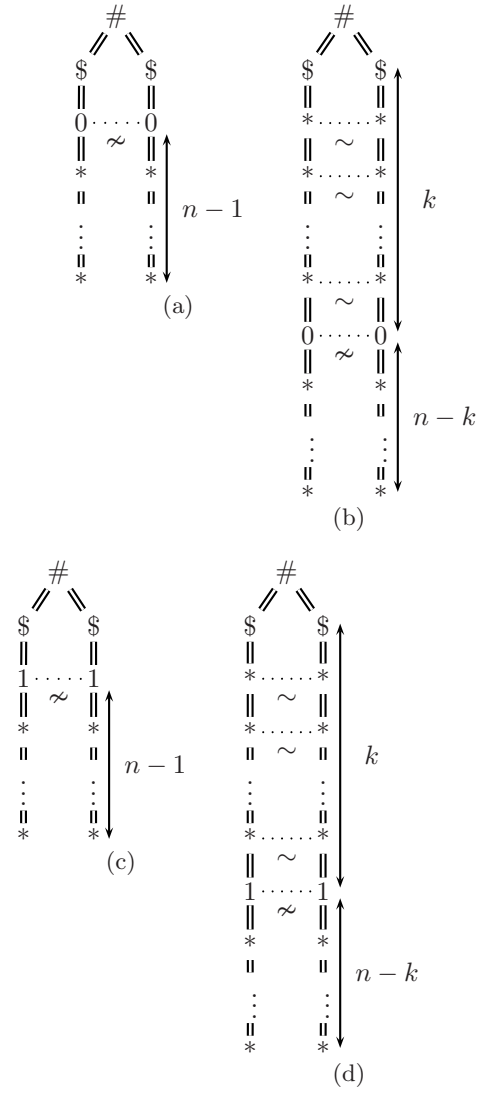
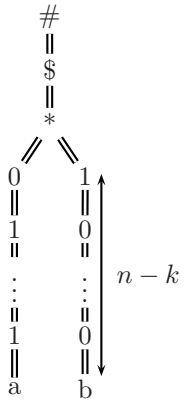


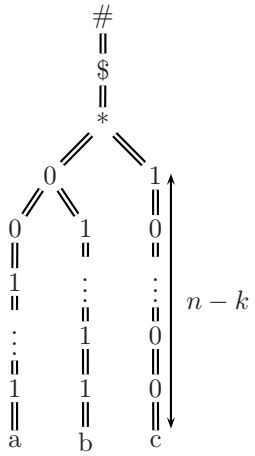
Fig. 9.

Then we have to ensure that transitions are simulated correctly. If we can talk about three consecutive positions of a configuration, it suffices to forbid bad behaviors of every such group of consecutive positions between two consecutive configurations.

We can distinguish two consecutive positions of a configuration. If two positions a and b satisfy a pattern of the following form, b is the successor of a in the configuration marked by \$.



Similarly, if position a, b and c satisfies a pattern of the following form, they are consecutive positions in the configuration.



In this way, we can construct patterns corresponding to bad behaviors as in the example of Figure 10.

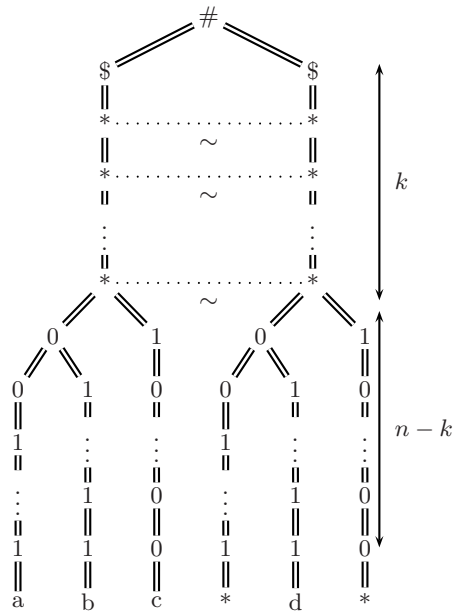


Fig. 10.

Taken together, we need a polynomial number of negated patterns to ensure data value properties and the coherence of transition between consecutive configurations.

We can build a DTD and a pattern formula of size polynomial in the size of the machine whose models are exactly the encodings of the accepting runs. □

B.3 Child fragment

Here we explain the upper bound part of Theorem 8:

Lemma 1. *The satisfiability problem in the presence of a DTD for BC*$(\sim, |)$ *is in* 2ExpTime.

Proof. First, we prove that a pattern formula φ of BC$(\sim, |)$ is satisfiable iff it is satisfiable by a data tree with a number of classes doubly exponential in the size of the formula. Then we show that we can recognize the data erasure of such a small models with an automaton of size doubly exponential in the size of the formula. Because emptiness of such automata is PTime, we have the 2ExpTime upper bound.

Consider a pattern formula φ and a data tree t that satisfies the formula. Consider a truth valuation of φ that is satisfied by the data tree t. We denote by positive (resp. negative) patterns, the patterns of the formula that must be true (resp. false) in the valuation. Consider also a witness function associating nodes of the positive patterns with nodes of the tree t. We call *marked* the nodes of t which are associated with nodes of the positive pattern.

First notice that we can pump *horizontally*[3] into the tree t with respect to the DTD to bound the arity of the tree by r, where r is polynomial in the size of the DTD and the pattern formula. We obtain a new tree t_r of polynomial arity. By construction, the tree respects the DTD. Because it still contains the marked positions, it satisfies the positive patterns. Because, patterns do not allow sibling edges, and we have pumped only in the width, the tree still respects the negated patterns.

From this new model t_r, we can construct a model t_{rc} with the same data erasure but a number of classes bounded by a polynomial constant of the form $m = p + r^d$. The number d is the sum of the depth of negated patterns and p represents the sum of the sizes of the patterns.
More precisely, we can rename data values top-down so that the data relation between all nodes is preserved in every subtree of depth d. Because of positive patterns, we do not rename classes corresponding to a marked position. Renaming data values does not change anything to the DTD membership. Moreover, since we do not rename the marked positions, the tree t_{rc} contains at lea st a witness for each positive pattern. Finally, as negated patterns are of depth smaller than d, the new data tree respects the negated patterns. Thus, the tree t_{rc} is still a model of the formula.

This ends the first part of the proof.

We can build a finite automaton over the alphabet $\Sigma \times M$ that recognizes exactly the models of φ and DTD of arity less than r and at most m classes labeled with M letters. This automaton is of size double exponential in size of φ and DTD and can be built in time 2ExpTime. Emptiness of finite tree automaton is PTime.

The satisfiability problem is thus 2ExpTime. □

Lemma 2. *The satisfiability problem in the presence of a DTD for BC*$(\sim, |)$ *is* AExpSpace*-hard.*

Proof. This lower bound result is obtained by a coding of accepting runs of AExpSpace Turing machines. Recall that the complexity classes AExpSpace and 2ExpTime are equal. We explain how to encode a run of an ExpSpace Turing machine into a data tree. Then we give the idea how to extend the model to AExpSpace Turing machines.

Consider a Turing machine of space 2^n and a input word x of length n. A run (see Figure below) is encoded in a data tree containing a branch labeled

[3] We cannot pump vertically because of negated patterns containing some child predicate. Thus if we pump vertically, we can introduce witnesses for a negated pattern.

by the sequence of transitions. In each node of this branch, we plug a subtree corresponding to the configuration. These configuration subtrees are defined as in the proof of Theorem 9.

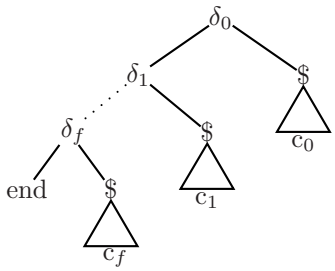


Fig. 11.

We can build a DTD and a pattern formula from BC($\sim$, $|$) such that a data tree is a model on the pattern formula and respects the DTD iff it is the encoding of an accepting run of the machine. The DTD constrains the shape of the tree. In particular it ensures that the sequence of transitions respects the machine's rules (succession of control states, initial and final configurations). As in the proof of Theorem 9, we use the fact that patterns can compare positions of a tree of polynomial depth. We construct patterns close to the one used in the proof of Theorem 9, and a pattern formula ensuring that the evolution of the configurations is correct. Using the child predicate it's possible to talk about consecutive configurations in the branch.

These two constructions can be extended to AExpSpace Turing machines. We encode runs by branching the coding of ExpSpace Turing machine runs. The encoding of the run is a tree labeled by the sequence of transitions and in each node of this branch, we plug the $\mathcal{C}$-subtrees of the corresponding configuration of the machine. The pattern formula and DTD can be built with the same idea. □

B.4 Bounded-depth Satisfiability

In this part, we prove Theorem 10 and Theorem 11:
Fixed bounded depth (Theorem 10)

If d is fixed, bounded depth satisfiability in the presence of a DTD for BC($\sim$, $\|$, $|$) is Σ_2-complete.

Lemma 3. *If d is fixed, bounded depth satisfiability in the presence of a DTD for BC($\sim$, $\|$, $|$) is Σ_2.*

Proof.

Consider a DTD L and a formula φ and a truth valuation of the formula that is satisfiable with respect to L. This valuation is satisfiable by a data tree of size polynomial in the size of the formula (exponential in the size of bound d). Consider a model of this valuation whose data erasure belongs to L. For each positive pattern of the valuation let mark the positions of a witness. Ignoring data values, as in classical regular tree languages, we can pump horizontally any part of the tree that does not contain a marked position with respect to L. By pumping we reduce the size of those parts to be of rank polynomial in L. The depth of the tree is fixed. Because there are a polynomial number of such positions, we obtain a new tree of polynomial size in φ and L. The data erasure of the new tree is still in L, and it still contains the witness positions. Moreover, because the pattern do not allow horizontal constraints, if the original tree does not contain any forbidden pattern, the new tree does not either. This new tree is still a model of the formula. □

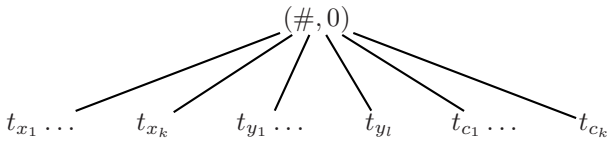
Lemma 4. *Bounded depth satisfiability in the presence of a DTD for BC*$(\sim, \|, |)$ *is* Σ_2*-hard for any fixed bound greater than three.*

Proof. This lower bound result is obtained by a reduction of QSAT2 to the satisfiability problem of BC$(\sim, \|)$ in the presence of a DTD. QSAT2 problem is the satisfiability problem of quantified propositional Boolean formulas with quantifier depth two (ie formula of the form $\exists X \forall Y \varphi$ where φ is in 3-DNF).

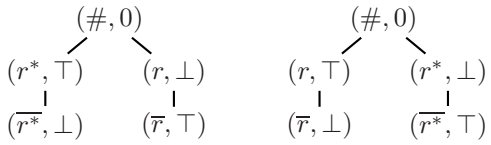
Consider a formula $\exists X \forall Y \varphi$ where φ is in 3-DNF. The idea of the proof is the following. There exists a set of data trees S and pattern ${P^X}_\varphi$ such that one of the data trees satisfies the negated pattern iff $\exists X \forall Y \varphi$ is satisfiable. This set of data trees S can be described as the models of pattern formula that respect a special DTD.

In the following we describe this set S, and the form of the pattern ${P^X}_\varphi$.

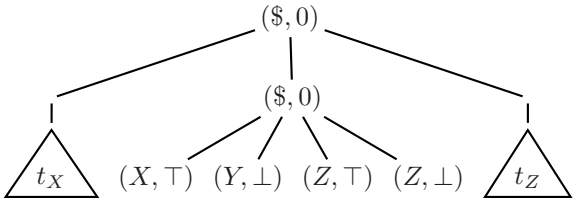
Intuitively, every data tree form S corresponds to the formula φ with a marked valuation of X variables. The structure of a data trees from S is similar to the one of the tree t_φ described in the proof of Proposition 4. The alphabet is $\Sigma = \{r, \overline{r}, r^*, \overline{r}^*, X, Y, Z, \#, \$\}$. A data tree in S contains exactly the six classes 0,1,2,3,$\top$ and $\bot$. It also contains all the possible values for each variable. Here we consider in disjunctive normal form, the tree contains all the possible values for each triplet of literals for a clause to be *false*. Additionally, we impose that each data tree contains a marked valuation of X variables. The structure is depicted in Figure 12.



(a) The form of a data tree of S



(b) The two possible subtrees t_x



(c) The subtree t_c with t_Y corresponding to Y false.

Fig. 12. Data trees of S

The t_y subtrees are identical to t_v subtrees in the proof of Proposition 4. The t_x subtrees are very similar. The only difference is that we mark one of the two valuations. Either the variable is true and is negation is false or it is false and its negation is true. A clause is viewed here as the conjunction of three literals, say X, Y, and Z. Each subtree t_c is still formed by three subtrees (t_X, t_Y and t_Z), each of them now representing one of the tree disjoint possibilities for a clause being false (1) X is false, or (2) X is true and Y is false, or (3) X and Y are true and Z is false. Figure 12 shows t_c with the subtree t_Y corresponding to Y false. Notice that the depth of the tree is three.

The pattern ${P^X}_\varphi$ is similar to the pattern P_φ. Thus we do not describe if formally. It selects a valuation per variable and per clause. Its structure makes sure that only one valuation per variable and per literal is selected. The only difference is that the valuation selected for X variables must be the marked one. The data constraints of the pattern ensure the consistency of the selection between variable part and clause part as in the proof of Proposition 4. Informally, the pattern describes a valuation of Y variables such that the formula is false with the marked valuation of X.

We have the following property: if a data tree of S satisfies the pattern ${P^X}_\varphi$, the marked valuation of X variables is not a solution for our instance of 2QSAT problem.

It is easy to see that we can describe the set S of data trees with a simple DTD that constrains the structure and node labels of a tree. A single pattern can constrain data (in)equality between all the nodes.

Hence, given a QSAT2 instance $\exists X \forall Y \varphi$, we can construct a DTD and a pattern formula such that the pattern formula is satisfiable with respect to the DTD iff the instance of QSAT2 has a solution. The pattern formula is simple as it is the conjunction of a pattern and a negated pattern. □

Fixed bounded depth (Theorem 11)

If d is part of the input, bounded depth satisfiability in the presence of a DTD for BC($\sim, \|, |$) is NExpTime-hard.

Proof. The proof of the lower bound is similar to the one of Theorem 9.

The upper bound comes from a small model property. Given a model of the formula, there exists a model of size exponential in the size of the input. Consider a pattern formula φ and a DTD L. Consider a model t of φ that respect the DTD L. Mark positions of one witness for each pattern of φ satisfied by t. Ignoring data value, as in classical regular tree language, we can pump horizontally w.r.t. the DTD any part of the tree that does not contain one of the marked positions. This way, we reduce the rank of the tree to be polynomial into the size of L and patterns of the formula φ. The depth is bounded by definition. The data erasure of the new tree still belongs to L. The new tree contains all the previous witnesses. And because we patterns do not allow sibling edges, if the original tree does not contain a pattern, the new tree does not either. Thus, the new tree is still a model of the formula.

A guess and check algorithm gives the upper bound. Guess a model, then evaluate the pattern on the tree and check the validity of the valuation. To evaluate the pattern, test each possible witness in the tree. Each pattern has a finite number of nodes so there is an exponential number of possible witnesses to test. The test is NP in the size of the model, which gives us the NExpTime upper bound. □

B.5 Positive Fragment

Here we show Proposition 13:

The satisfiability problem of a pattern formula that does not allow negation of pattern in the presence of a DTD is in NP-complete.

Notice this proof is very similar to the proof of Theorem 4.5 in [5].

Proof.

The lower bound is very easy. It can be obtain for example from the proof of Proposition 4, with the same reduction from 3SAT: the resulting pattern formula is a conjunction of two patterns. It is also be obtain by using other 3SAT as in coding [5] even without data value.

The upper bound proof is very similar to the proof of Theorem 4.5 in [5]. It is more difficult than the lower bound, mainly because even a simple DTD can force

every model to be of exponential size. However, the absence of negative patterns allows us to concentrate on the positions used as witnesses for the patterns. This allows us to guess only a small part of a model (of polynomial size) and check that it can be extended to a valid tree (without constructing it).

If a pattern formula from $\mathrm{BC}^+(\sim, |, \|)$ has a model, there is a model of depth polynomial in the size of the DTD and the formula. Indeed, considering a model with a witness for each pattern, we can pump vertically with respect to the DTD in all subtrees that do not contain any witness position for a pattern of the formula. We obtain a new model of depth polynomial in the size of the DTD and the pattern formula.

In a model with polynomial depth, we now consider the minimal subtree containing one witness for each pattern. This subtree is of polynomial size.

The last remaining problem is to check whether we can build, from the data erasure of this tree, a tree satisfying the DTD. In order to do that, we guess the states labeling the nodes and transitions of the subtree in a successful run of the automaton corresponding to the DTD. We only have to check if it is possible to extend the tree into a tree satisfying the DTD with the guessed transitions. This takes polynomial time. We only have to check for each node whether the Boolean combination formula corresponding to a transition can be fulfilled by adding nodes to the tree. As there are no forbidden patterns, this padding cannot cause problems w.r.t. the pattern formula.

An NP algorithm consist of (1) guessing tree of polynomial size with witness positions and transitions (2) checking whether witnesses are correct and whether the tree can be extended w.r.t the DTD. □