Satisfiability of Constraint Specifications on XML Documents *

Marisa Navarro[1], Fernando Orejas[2], and Elvira Pino[2]

[1] Universidad del País Vasco (UPV/EHU), San Sebastián, Spain
marisa.navarro@ehu.es
[2] Universitat Politècnica de Catalunya, Barcelona, Spain
{orejas,pino}@cs.upc.edu

Abstract. Jose Meseguer is one of the earliest contributors in the area of Algebraic Specification. In this paper, which we are happy to dedicate to him on the occasion of his 65th birthday, we use ideas and methods coming from that area with the aim of presenting an approach for the specification of the structure of classes of XML documents and for reasoning about them. More precisely, we specify the structure of documents using sets of constraints that are based on XPath and we present inference rules that are shown to define a sound and complete refutation procedure for checking satisfiability of a given specification using tableaux.

1 Introduction

The aim of our work is to study how we can specify classes of XML documents and how we can reason about them. Currently, the standard specification of classes of XML documents is done by means of DTDs or XML Schemas. In both cases, we essentially describe the abstract syntax of the class of documents and the types of its attributes. This is quite limited. In particular, we may want to state more complex conditions about the structure of documents in a given class or about their contents. For example, with respect to the structure of documents, we may want to state that if an element includes an attribute with a given content, then these documents should not include some other element. Or, with respect to the contents of documents, we may want to express that the value of some numeric attribute of a certain element is smaller than the value of another attribute of a different element.

In this paper, we concentrate on the specification of the structure of documents, not paying much attention to their contents. In this sense, we present an abstract approach for the specification of (the structure of) classes of XML documents using sets of constraints that are based on XPath [17, 21] queries, as given in [11], using the concept of *tree patterns*. Roughly, a tree pattern describes a basic property on the structure of documents. Its root represents the root of documents. Nodes represent elements that

* This work has been partially supported by funds from the Spanish Ministry for Economy and Competitiveness (MINECO) and the European Union (FEDER funds) under grant COMMAS (ref. TIN2013-46181-C2-1-R, TIN2013-46181-C2-2-R) and from the Basque Project GIU12/26, and grant UFI11/45.

must be present on the given documents and their labels represent their contents, i.e. the names of elements and their value, if any. A wildcard (the symbol $*$), means that we don't know or we don't care about the contents of that element. Finally, single edges represent parent/child relations between elements, while double edges represent a descendant relationship between elements. Again, if any of these two relations is included in a tree pattern, then it should also be included in the documents satisfying that property. For instance, on the left of Fig. 1 we show a tree pattern p describing documents $\mathcal{D}$ whose root node is labelled with a, some child node of the root node in $\mathcal{D}$ is labelled b, and some descendant node of the root node in $\mathcal{D}$ has two child nodes labelled c and d, respectively.

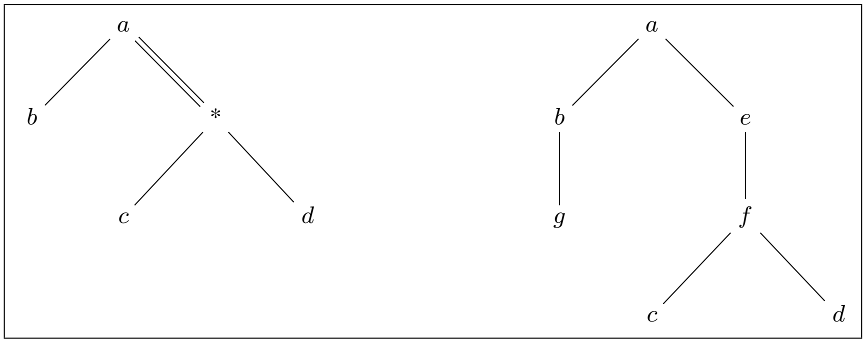

Fig. 1. A tree pattern and a document satisfying the pattern

Similarly, we represent, in an abstract way, XML documents using the same kind of trees. The difference between a document and a tree pattern is that a document does not include double edges or wildcards. For example, on the right of Fig. 1 we show a document that satisfies the pattern on the left. In particular, we may see that the root of the document is labelled by a. Moreover, that root has a child node labelled b and a descendant node (the element labelled f) that has two child nodes labelled c and d, respectively.

We consider three kinds of (atomic) constraints. The first one, called *positive constraints*, are tree patterns. The second one are *negative constraints*, $\neg p$, where p is a tree pattern, expressing that documents should not satisfy p. Finally, the third sort of constraint are *conditional constraints*, written $\forall(c : p \to q)$, where both p and q are tree patterns. Roughly speaking, these constraints express that if a document satisfies p then it must also satisfy q. Moreover, these constraints can be combined using the connectives $\wedge$ and $\vee$. These kinds of constraints are similar to the graph constraints studied in [15, 16] in the context of graph transformation. Nevertheless, the application of the ideas in [15, 16] to our setting is not trivial, as discussed in Sec. 3.

Obviously, there are conditions on the structure of XML documents that are not expressible using the kind of constraints studied in this paper. However, our experience in the area of graph transformation [15, 16] shows that, in practice, these constraints are sufficient in most cases. Nevertheless, we believe that the ideas presented here can be extended to a class of XML constraints, similar to the class of nested graph conditions that has been shown equivalent to first-order logic of graphs [6]. However, we also believe that this extension is not straightforward.

Since our aim is to be able to reason about these specifications, we present inference rules that are shown, by means of tableaux, to define a sound and complete refutation procedure for checking satisfiability of a given specification.

The paper is organized as follows. Section 2 contains some basic notions and notational conventions we are going to use along the paper. Section 3 introduces the three kinds of constraints that we use as literals of the clauses in a specification. In Sec. 4 we present our tableau method for reasoning about our constraints, and in Sec. 5 we show its soundness and completeness. Finally, in Sec. 6 and 7 we discuss related work and provide some conclusions.

2 Basic Definitions and Notation

In this section we introduce some basic concepts and notations, as well as some definitions and properties on patterns that will be required in the paper.

2.1 Documents and Patterns

As we have seen in the introduction, we consider a *document* as a kind of unordered and unranked tree with nodes labelled from an infinite alphabet Σ and whose edges represent a parent/child relation between nodes. The symbols in Σ represent the element labels, attribute labels, and values that can occur in documents. By considering that the trees are unordered and unranked, the subtrees can commute (the "sibling ordering" is irrelevant), and there are no restrictions on the number of children a node can have.

As also seen, patterns describe properties on the structure of documents and are also represented by trees. However, there is the special label $*$, representing the *wildcard*, and there are two kinds of edges: single and double edges. Patterns (and documents) can be represented textually using the following format: A pattern p with root labelled a and subtrees $p_1, \ldots, p_n$ will be textually written $p = a(!p_1) \ldots (!p_n)$ where each p_i is recursively written in the same format, and ! being / or // to indicate the edge from the root to each subtree p_i. Some parenthesis can be omitted in the case of having only one subtree. For instance, the pattern given in Fig. 1 can be textually written $a(/b)(// * (/c)(/d))$. Similarly, the document in the same Fig. 1 is textually written $a(/b/g)(/e/f(/c)(/d))$.

However, even if the documents and the patterns that we would write would always be finite, in our paper we need to deal with infinite documents and patterns. The reason, is that (as often done), given a specification for a class of documents, we will consider that the specification is consistent if there exist documents that satisfy it, even if these documents are infinite. In this sense, one might consider that the results shown in Sect. 5 are not fully adequate, in the sense that we would not have proved the completeness of our proof rules with respect to the class of finite documents.

For this reason, we need a more precise definition of what documents and patterns are. In particular, we define them as follows:

Definition 1 (Patterns and Documents). *Given a signature Σ, a* pattern p on Σ *is a 5-tuple $p = (Nodes_p, root_p, Label_p, Edges_p, Paths_p)$ where, $Nodes_p$ is a set of nodes, $root_p \in Nodes_p$ is the root node of p, $Label_p : Nodes_p \to \Sigma \cup \{*\}$ is the labeling function, and $Edges_p, Paths_p \subseteq Nodes_p \times Nodes_p$ are two relations representing edges and paths between nodes in p, such that the following conditions are satisfied:*

1. *$Edges_p$ and $Paths_p$ are irreflexive and acyclic (i.e. there are no nodes n, n' such that $\langle n, n' \rangle$ and $\langle n', n \rangle$ are both in $Edges_p$ or $Paths_p$).*
2. *$Paths_p$ is transitive and includes $Edges_p$.*
3. *There is no node n such that $\langle n, root_p \rangle$ is in $Edges_p$ or $Paths_p$.*
4. *For any other node $n \neq root_p$ in $Nodes_p$, $\langle root_p, n \rangle \in Paths_p$. Moreover, if $\langle n', n \rangle$ and $\langle n'', n \rangle$ are both in $Paths_p$, then either $\langle n'', n' \rangle$ or $\langle n', n'' \rangle$ are in $Paths_p$.*

Then, a document $\mathcal{D}$ on Σ *can be defined as a special kind of pattern without nodes labelled with $*$, that is, $Label_{\mathcal{D}} : Nodes_{\mathcal{D}} \to \Sigma$, and, such that, in addition it satisfies the following condition:*

5. *For every pair of nodes $n, n' \in Nodes_{\mathcal{D}}$, if $\langle n, n' \rangle \in Paths_{\mathcal{D}}$, then*
   - *$\langle n, n' \rangle \in Edges_{\mathcal{D}}$, or*
   - *there exist $n_1, n_2 \in Nodes_{\mathcal{D}}$ such that $\langle n, n_1 \rangle, \langle n_2, n' \rangle \in Edges_{\mathcal{D}}$ and, either $n_1 = n_2$ or $\langle n_1, n_2 \rangle \in Paths_{\mathcal{D}}$.*

P_Σ and D_Σ will denote, respectively, the set of all patterns and the set of all documents on Σ.

Intuitively, the above definition can be easily explained. The relation $\langle n, n' \rangle \in Edges_p$ represents the existence of an edge $/$ between n and n' in the given pattern or document, and $\langle n, n' \rangle \in Paths_p$ represents that there is a path consisting of edges $/$ or $//$ (in the case of patterns) or just $/$ (in the case of documents) between n and n'. Conditions 1-4 ensure that our patterns and documents are trees. Finally, Cond. 5 ensures that if $\langle n, n' \rangle \in Paths_{\mathcal{D}}$ then there is a finite or infinite path, consisting only of edges $/$, between n and n'. It is easy to see that, in the case where the given set of nodes is finite, our definition of patterns and documents would be equivalent to other notions of (finite) trees. In particular, for finite documents, Condition 5 is equivalent to saying that $Paths$ is the transitive closure of $Edges$.

One could think that the second part of Cond. 5 could be simplified as follows:

– there exists $n_1 \in Nodes_{\mathcal{D}}$ such that $\langle n, n_1 \rangle \in Edges_p$ and $\langle n_1, n' \rangle \in Paths_p$.

However, both conditions are not equivalent. In particular, our Cond. 5 would exclude infinite paths like $n/n_1/n_2/\ldots/n_k/\ldots$, where for every i, $\langle n_i, n' \rangle \in Paths_p$, which would be allowed by the simpler condition. That is, an infinite path for $\langle n, n' \rangle \in Paths_p$ cannot consist of an infinite sequence $n/n_1/n_2/\ldots/n_k/\ldots$ *approaching* n'. Instead, our infinite paths must consist of two infinite sequences $n/n_1/n_2/\ldots/n_k/\ldots$ and $\ldots/n'_j/\ldots/n'_2/n'_1/n'$, where for every i, i', $\langle n_i, n'_{i'} \rangle \in Paths_p$.

For example, consider again the pattern and the document in Fig. 1. Abusing of notation, let us identify nodes with labels. Then, for the pattern $p = a(/b)(//{*}(/c)(/d))$ and the document $\mathcal{D} = a(/b/g)(/e/f(/c)(/d))$, we have:

$Edges_p = \{\langle a, b \rangle, \langle *, c \rangle, \langle *, d \rangle\}$

$Paths_p = \{\langle a,b\rangle, \langle a,*\rangle, \langle a,c\rangle, \langle a,d\rangle, \langle *,c\rangle, \langle *,d\rangle\}$
$Edges_{\mathcal{D}} = \{\langle a,b\rangle, \langle b,g\rangle, \langle a,e\rangle, \langle e,f\rangle, \langle f,c\rangle, \langle f,d\rangle\}$
$Paths_{\mathcal{D}} = \{\langle a,b\rangle, \langle a,e\rangle, \langle a,g\rangle, \langle a,f\rangle, \langle a,c\rangle, \langle a,d\rangle, \langle b,g\rangle, \langle e,f\rangle, \langle e,c\rangle, \langle e,d\rangle, \langle f,c\rangle,$
$\langle f,d\rangle\}$

For the sake of readability, from now on we will omit the signature Σ. Moreover, we will write n/n' instead of $\langle n,n'\rangle \in Edges_p$, and $n//n'$ instead of $\langle n,n'\rangle \in Paths_p$. Notice that, in our simplified notation, the symbol // is overloaded to mean a kind of edge in patterns but also, the relation defining paths in patterns and documents. However, it is easy to distinguish both uses from the context since, in the first case, we will usually refer to "an edge $//$". If some ambiguity could persist then, we will use $//^d$ to denote direct relation between nodes. That is, given $n_1, n_2 \in Nodes_p$ $n_1//^d n_2$ if not n_1/n_2 but, $n_1//n_2$ such that there does not exist $n \in Nodes_p$ with $n_1//n$ and $n//n_2$. Nevertheless, for simplicity, in the examples in the rest of the paper we will use the textual writing for patterns and documents, so that, in those expressions the symbol $//$ always will stand for edges, that is, the direct relation $//^d$.

2.2 Pattern Morphisms and Pattern Models

Morphisms are very important in our work. A document satisfies a pattern if we can identify the structure of the pattern in the document. Formally, we do this by means of morphisms. In addition, we also use a special kind of morphisms to relate the premise and the conclusion in conditional constraints. We define the notion of morphism between two patterns, since documents are a special case of patterns. Then, the same definition applies to morphisms between documents or between patterns and documents. As said, the latter case will be used to define which documents are the models of a pattern. That is, from a logical point of view, we can see patterns as formulae, documents as structures and morphisms defining a notion of pattern satisfaction.

Definition 2 (Morphisms). *Given two patterns $p, q \in P$, a* morphism $h : p \to q$, *from p to q, is a function $h : Nodes_p \to Nodes_q$ satisfying the following conditions:*

- *Root-preserving: $h(root_p)$=$root_q$;*
- *Label-preserving: For each $n \in Nodes_p$, $Label_p(n)$=$*$ or $Label_p(n)$=$Label_q(h(n))$;*
- *Edge-preserving: For each $n_1, n_2 \in Nodes_p$, if n_1/n_2 then, $h(n_1)/h(n_2)$;*
- *Path-preserving: For each $n_1, n_2 \in Nodes_p$, if $n_1//n_2$ then, $h(n_1)//h(n_2)$;*

As usual, a monomorphism is an injective morphism. $\mathbf{P_\Sigma}$ and $\mathbf{D_\Sigma}$ denote, respectively, the category of patterns and its subcategory of documents on Σ.

Definition 3 (Models). *Given a pattern $p \in P$ and a document $\mathcal{D} \in D$, we say that $\mathcal{D}$* satisfies *p, denoted $\mathcal{D} \models p$, if there exists a monomorphism from p to $\mathcal{D}$. The set of models of a pattern p is the set of documents satisfying p, that is, $Mod(p)$ = $\{\mathcal{D} \in D \mid \mathcal{D} \models p\}$.*

In Fig. 2 there is an example of a monomorphism $h : p \to \mathcal{D}$ from the pattern p = $a(/b)(//*(/c)(/d))$ to the document $\mathcal{D}$ = $a(/e/f(/c)(/d))(/b/g)$. The morphism h is

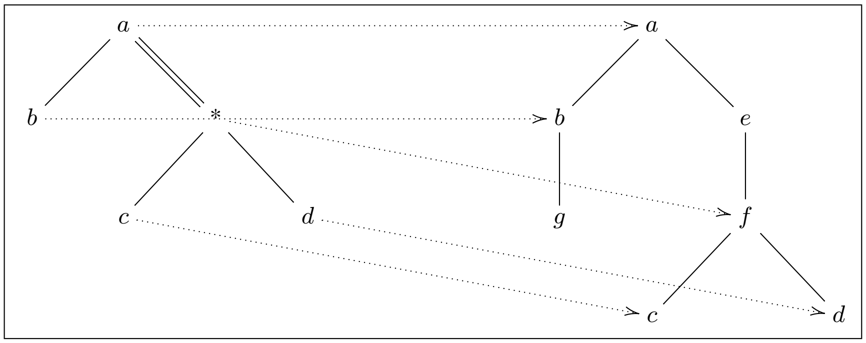

Fig. 2. A pattern p, a document $\mathcal{D}$ and a monomorphism $h : p \rightarrow \mathcal{D}$

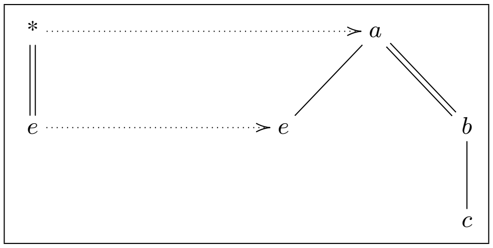

Fig. 3. A monomorhism $h : p \rightarrow q$ between two patterns

drawn with dotted arrows. We can see that $\mathcal{D}$ satisfies p because its root is labelled with a, it has a child node labelled b, and it has a descendant node (in the example labelled with f) with two child nodes labelled with c and d respectively. In Fig. 3 there is an example of a monomorphism $h : p \rightarrow q$ from the pattern p = $*//e$ to the pattern q = $a(/e)(//b/c)$. The monomorphism h is drawn with dotted arrows. The existence of such monomorphism implies that all models of q are also models of p.

The following proposition relates monomorphisms and models for two patterns.

Proposition 1. *Given two patterns $p, q \in P$:*

- *If there exists a monomorphism $h : p \rightarrow q$ then $Mod(q) \subseteq Mod(p)$.*
- *$Mod(q) \subseteq Mod(p)$ does not imply that there is a monomorphism $h : p \rightarrow q$.*

Proof. For the first claim, let $\mathcal{D}$ be a document in $Mod(q)$, then there exists a monomorphism f from q to $\mathcal{D}$. Then the composition $f \circ h$ is a monomorphism from p to $\mathcal{D}$ and therefore the document $\mathcal{D}$ is also a model for p. The second claim can be shown with an example in [11] .

3 Constraints, Clauses and Specifications

As said in the Introduction, following [15, 16] we consider three kinds of constraints: positive, negative and, conditional constraints. The underlying idea of our constraints is that they should specify that certain patterns must occur (or must not occur) in a given document. For instance, the simplest kind of constraint, p, specifies that a given document $\mathcal{D}$ should satisfy the pattern p. Obviously, $\neg p$ specifies that a given document $\mathcal{D}$ should not satisfy p. A more complex kind of constraint is of the form $\forall(c : p \rightarrow q)$ where c is a prefix morphism, which means that q is a pattern that extends p. Roughly speaking, this constraint specifies that whenever a document $\mathcal{D}$ satisfies the pattern p it should also satisfy the extended pattern q (see Def. 6 below).

However, translating the ideas in [15, 16] to our setting is not trivial, mainly for two reasons. On the one hand, in [15, 16] models and formulas are both graphs, while in our setting models are documents and formulas are patterns. This difference adds some complication to our setting. In particular, we have solved the problem by defining documents and patterns in such a way that documents are a special case of patterns. This has implied to include explicitly the $paths$ relation in the definition of documents. On the other hand, and most importantly, we deal with patterns that are trees having edges of type //, but the related notion of “path" is not considered for graph constraints in [15, 16]. Actually, in the logic defined in [15, 16] or in the more general one defined in [6], the existence of paths is a second order notion [7].

3.1 Constraints and Clauses

Before defining our three kinds of constraints, we must define prefix morphisms.

Definition 4. *Given two patterns p and q, a* prefix morphism *from p to q is a monomorphism* $c : Nodes_p \rightarrow Nodes_q$ *that satisfies the following conditions:*

- *Root-preserving:* $c(root_p){=}root_q$*;*
- *Label -identity: For each* $n \in Nodes_p$*,* $Label_p(n){=}Label_q(c(n))$*;*
- *Edge-identity: For each* $n_1, n_2 \in Nodes_p$*,* n_1/n_2 *if, and only if,* $c(n_1)/c(n_2)$*;*
- *Path-identity: For each* $n_1, n_2 \in Nodes_p$*,* $n_1//^d n_2$ *if, and only if,* $c(n_1)//^d c(n_2)$*;*

Recall that $//^d$ stands for direct $//$-relation in patterns, that is, $//$-edges. We will simply write $c : p \rightarrow q$ and we will say that p is a prefix of q. Not every monomorphism is a prefix morphism. See for instance that the monomorphism in Fig. 3 is not a prefix morphism since it violates "Label-identity", "Edge-identity", and "Path-identity".

Definition 5. *Given a pattern p, p denotes a* positive constraint *and* $\neg p$ *denotes a* negative constraint*. A* conditional constraint *is denoted* $\forall(c : p \rightarrow q)$ *where p and q are patterns and* $c : p \rightarrow q$ *is a prefix morphism.*

A clause α *is a finite disjunction of literals* $\ell_1 \vee \ell_2 \vee \ldots \vee \ell_n$*, where, for each* $i \in \{1, \ldots, n\}$*, the literal* ℓ_i *is a (positive, negative or conditional) constraint. The empty disjunction is called the* empty clause *and it can be represented by* $FALSE$*.*

Satisfaction of clauses is inductively defined as follows.

Definition 6. *A document $\mathcal{D} \in D$ satisfies a clause α, denoted $\mathcal{D} \models \alpha$, if it holds:*

- *$\mathcal{D} \models p$ if there exists a monomorphism $h : p \rightarrow \mathcal{D}$;*
- *$\mathcal{D} \models \neg p$ if $\mathcal{D} \not\models p$ (that is, if there does not exist a monomorphism $h : p \rightarrow \mathcal{D}$);*
- *$\mathcal{D} \models \forall(c : p \rightarrow q)$ if for every monomorphism $h : p \rightarrow \mathcal{D}$ there is a monomorphism $f : q \rightarrow \mathcal{D}$ such that $h = f \circ c$.*
- *$\mathcal{D} \models \ell_1 \vee \ell_2 \vee \ldots \vee \ell_n$ if $\mathcal{D} \models \ell_i$ for some $i \in \{1, \ldots, n\}$.*

Let us see what satisfaction of a conditional constraint means. First recall that in a conditional constraint $\forall(c : p \rightarrow q)$ the pattern q is an extension of the pattern p so a document $\mathcal{D}$ will be a model of the conditional constraint if whenever $\mathcal{D}$ satisfies p, it also satisfies q. To be more precise, each part in the document $\mathcal{D}$ satisfying p must satisfy q. Consider, for instance, the conditional constraint $\forall(c : p \rightarrow q)$ with $p = *//a$, $q = *//a/b$ and c being the obvious prefix morphism from p to q. By Def. 6, a document satisfies this constraint if each node (descendant of the root) labelled a has a child node labelled b. Then the document $\mathcal{D} = g(/a/b)(/a/h)$ does not satisfy the constraint. In fact, for the monomorphism $h : p \rightarrow \mathcal{D}$ that applies the node a in p into the second node a in $\mathcal{D}$, there does not exist a monomorphism $f : q \rightarrow \mathcal{D}$ such that $h = f \circ c$. However, note that $\mathcal{D} \models q$. Therefore, from a logical point of view, we may notice that, in this framework, $\forall(c : p \rightarrow q)$ is not equivalent to $C = \neg p \vee q$.

3.2 Specifications

We assume that a specification $\mathcal{S}$ consists of a set of clauses. As said in the Introduction, our aim is to find a sound and complete refutation procedure for checking satisfiability of specifications consisting of clauses as defined above. Here we give an example of an unsatisfiable specification.

Example 1. Consider the specification $\mathcal{S} = \{C_1, C_2, C_3, C_4\}$ where $C_1 = (*//b) \vee (*//e)$, $C_2 = \forall(c_2 : *//b \rightarrow *(//b)(/e))$, $C_3 = \forall(c_3 : *//e \rightarrow *(//e)(/b))$, and $C_4 = \neg(*(/b)(/e))$.
Clause C_1 specifies that the document(s) must have a node labelled b or e; C_2 says that if the document has some node labelled b then its root must have a child node labelled e; similarly, C_3 says that if the document has some node labelled e then the root must have a child node labelled b; and finally, C_4 says that the root cannot have two children nodes labelled b and e. It is easy to test, for instance, that the document $\mathcal{D}_1$= $a(/b)(/f/e)$ satisfies C_1, C_3 and C_4 but $\mathcal{D}_1 \not\models C_2$. Similarly, the document $\mathcal{D}_2 = a/e$ satisfies C_1, C_2 and C_4 but $\mathcal{D}_2 \not\models C_3$. There is no document satisfying all clauses in $\mathcal{S}$.

3.3 Superposition of Patterns

In this section we introduce two operations on patterns that can be seen as a way of pattern deduction, which will be used for obtaining new clauses from a specification. Note, for instance, that if a document $\mathcal{D}$ satisfies both the patterns a/b and a/c then its root, labelled a, must have two children nodes labelled b and c, therefore we can trivially deduce that $\mathcal{D}$ must also satisfy the pattern $a(/b)(/c)$. But not always a single pattern can express the conditions of two patterns: If $\mathcal{D}$ satisfies both the patterns $a/b/e$ and $a/b/c$, then it must be deduced that $\mathcal{D}$ satisfies one of the patterns obtained by superposing both patterns, what means, in this example, that $\mathcal{D}$ must satisfy either the pattern $a/b(/e)(/c)$ or the pattern $a(/b/e)(/b/c)$.

The two superposition operations on patterns will be denoted by the symbols $\otimes$ and $\otimes_{c,m}$ and they are formally introduced in the following definitions.

Given two patterns p_1 and p_2, the operation $p_1 \otimes p_2$ denotes the set of patterns that can be obtained by "combining" p_1 and p_2 in all possible ways.

Definition 7. *Given two patterns p_1 and p_2, $p_1 \otimes p_2$ is defined as the following set of patterns: $p_1 \otimes p_2$ = {$s \in P$ | there exist jointly surjective monomorphisms $inc_1 : p_1 \to s$ and $inc_2 : p_2 \to s$} where "jointly surjective" means that $Nodes_s = inc_1(Nodes_{p_1}) \cup inc_2(Nodes_{p_2})$.*

For instance, given the patterns $p_1 = a(/b/e)(//c)$ and $p_2 = a//b/x$, the set $p_1 \otimes p_2$ contains the two patterns: $s_1 = a(/b(/e)(/x))(//c)$ and $s_2 = a(/b/e)(//b/x)(//c)$. Each one corresponds with a way of combining p_1 and p_2; the nodes labelled b are shared in s_1 while there are two different nodes b in $s2$.

The underlying idea is that all patterns s in $p_1 \otimes p_2$ must verify that every document that is a model of s must be a model of p_1 and a model of p_2. Conversely, every document that is a model of both p_1 and p_2 must be a model of some s in $p_1 \otimes p_2$. In some cases the set can be empty.

Proposition 2. *Given two patterns p_1 and p_2, the set of patterns $p_1 \otimes p_2$ is the empty set if and only if $root_{p_1}$ and $root_{p_2}$ have different labels in Σ.*

Proof. If the roots of p_1 and p_2 have different labels in Σ (for instance, a and b) then no combination s is possible since $inc_1 : p_1 \to s$ implies that the root of s must be labelled a and $inc_2 : p_2 \to s$ implies that the root of s must be labelled b.

Conversely, if the root s of p_1 and p_2 have the same label a (or if one of them is a and the other one is $*$) then the document s with root labelled a and whose set of subtrees is the union of the subtrees of p_1 and p_2 is an element in $p_1 \otimes p_2$. ■

Proposition 3 (Pair Factorization Property). *Given three patterns p_1, p_2, r, and two monomorphisms $f_1 : p_1 \to r$ and $f_2 : p_2 \to r$, there exists a pattern $s \in p_1 \otimes p_2$, with monomorphisms $inc_1 : p_1 \to s$ and $inc_2 : p_2 \to s$, and there exists a monomorphism $h : s \to r$ such that $h \circ inc_1 = f_1$ and $h \circ inc_2 = f_2$. In the particular case when r is a document, this property means that r is a model of s . Graphically:*

$$\begin{tikzcd}
p_1 \arrow[d, "inc_1"'] \arrow[dr, "f_1"] & \\
s \arrow[r, "h"] & r \\
p_2 \arrow[u, "inc_2"] \arrow[ur, "f_2"'] &
\end{tikzcd}$$

Proof. Since f_1, f_2 are monomorphisms, the root s of p_1 and p_2 cannot have different labels in Σ. Moreover, some pattern $s \in p_1 \otimes p_2$ holds this property. Then, we will have a well-defined morphism h if choose a pattern s such that, for every $m \in Nodes_{p_1}$ and $n \in Nodes_{p_2}$: if $f_1(m) = f_2(n)$ then $inc_1(m) = inc_2(n)$ and if $f_1(m)$ is an ancestor (respectively descendant) of $f_2(n)$, $inc_1(m)$ must not be a descendant (respectively ancestor) of $inc_2(n)$. ■

Given a pattern p_1, a prefix morphism $c : p_2 \rightarrow q$ and a monomorphism $m : p_2 \rightarrow p_1$, the operation $p_1 \otimes_{c,m} q$ denotes the set of patterns that can be obtained by combining p_1 and q in all possible ways, but sharing p_2.

Definition 8. *Given a pattern p_1, a prefix morphism $c : p_2 \rightarrow q$, and a monomorphism $m : p_2 \rightarrow p_1$, $p_1 \otimes_{c,m} q$ is defined as the following set of patterns: $p_1 \otimes_{c,m} q = \{s \in P \mid$ there exist jointly surjective monomorphisms $inc_1 : p_1 \rightarrow s$ and $inc_2 : q \rightarrow s$ such that $inc_1 \circ m = inc_2 \circ c\}$.*

For instance, given the patterns: $p_1 = a(/b/e)(//c/i)$, $p_2 = *//b$, and $q = *(//b//a)(//c/d)$, with the unique possible monomorphism $m : p_2 \rightarrow p_1$ and the unique possible prefix morphism $c : p_2 \rightarrow q$, the set $p_1 \otimes_{c,m} q$ contains the patterns $s_1 = a(/b(/e)(//a))(//c/i)(//c/d)$ and $s_2 = a(/b(/e)(//a))(//c(/i)(/d))$. Note that s_2 is similar to s_1 but with only one node labelled c.

The underlying idea is that all patterns s in $p_1 \otimes_{c,m} q$ must verify that every document $\mathcal{D}$ that is a model of s must be a model of p_1 and a model of q. However, such a document $\mathcal{D}$ is not necessarily a model of the conditional constraint $\forall(c : p_2 \rightarrow q)$. Conversely, every document that is a model of both p_1 and $\forall(c : p_2 \rightarrow q)$ must be a model of some s in $p_1 \otimes_{c,m} q$.

Notice that the set $p_1 \otimes_{c,m} q$ is always non-empty, since given a prefix morhism $c : p_2 \rightarrow q$ and a monomorphism $m : p_2 \rightarrow p_1$ we can always obtain a pattern s by extending the nodes in $m(p_2)$ as indicated by the function c. However, if c would be a monomorphism instead of a prefix morphism (i.e. if in the definition of conditional constraints we would have used arbitrary monomorphisms), the resulting set could be empty. Take, for instance, $c : p_2 \rightarrow q$, with $p_2 = a//b$ and $q = a/b$ (which is not a prefix morphism), and take $p_1 = a/e/b$. Although there is a monomorphism $m : p_2 \rightarrow p_1$, there is no pattern s obtained by combining p_1 and q sharing p_2.

4 Tableau-based Reasoning for XML-patterns

Analogously to tableaux for plain first-order logic reasoning [8], we introduce tableaux for dedicated automated reasoning for XML-document properties. In general, tableaux are trees whose nodes are literals. In our case, these literals are constraints of the form p, $\neg p$ or $\forall(c : p \rightarrow q)$.

Definition 9 (Tableau, branch). *A* tableau *is a finitely branching tree whose nodes are constraints. A* branch *B in a tableau T is a maximal path in T.*

The construction of a tableau for a specification S can be informally explained as follows. We start with a tableau consisting of the single node *true*. For every clause $\alpha = \ell_1 \vee \ldots \vee \ell_n$ in $\mathcal{S}$ we extend all the leaves in the tableau with n branches, one for each literal ℓ_i, as depicted in Fig. 5 (see the four first steps) for the specification S in Example 1. Then we continue the extension of each leaf by applying other tableau rules (on two literals in its branch). In Fig. 5 we show the tableau in the left hand side, and the rules applied in its construction in the right hand side.

Before defining the rules that build the tableaux associated to our specifications, we need to introduce some notation. Let p be a positive constraint in B, such that p contains an edge $//$ (that is, $n_1//^d n_2$ for some $n_1, n_2 \in Nodes_p$). Let $prefix(n_1)$ denote the path from $root_p$ to the node n_1, and $hang(n_2)$ the subtree of p hanging from the node n_2; we write $p[n_1//hang(n_2)]$ to highlight the edge $//$ from node n_1 to node n_2 in p. Then $p[n_1/hang(n_2)]$ denotes the pattern obtained by replacing $//$ by $/$. In addition, $p[n_1 \leftarrow]$ denotes that the subtree $hang(n_2)$ has been pruned from p, and $p[n_1 \leftarrow /A]$ (equivalently $p[n_1 \leftarrow //A]$) denotes that the pattern A is hanged as a subtree of node n_1 in p, where $/$ (equivalently $//$) is the edge from n_1 to $root_A$.

For instance, given the pattern $p = e(/i)(/a(/b)(//c(/d)(/j)))$ (see Fig. 4) with an edge $//$ from the node n_1 labelled a to the node n_2 labelled c, we have that $prefix(n_1) = e/a$; $hang(n_2) = c(/d)(/j)$; $p[n_1 \leftarrow] = e(/i)(/a/b)$; and $p[n_1 \leftarrow /A] = e(/i)(/a(/b)(/s/k))$ when hanging, for instance, the pattern $A = s/k$.

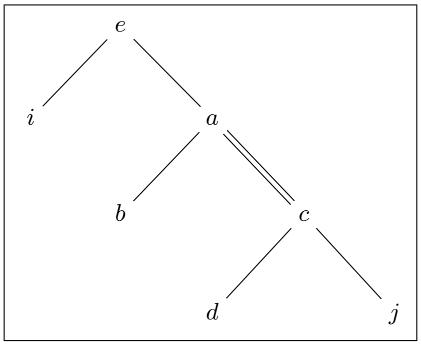

Fig. 4. The pattern $p = e(/i)(/a(/b)(//c(/d)(/j)))$

Now, the tableau rules that are specific for our logic are the following ones:

Definition 10 (Tableau rules). *Given a specification $\mathcal{S}$, a tableau for $\mathcal{S}$ is either a tree consisting of the single node* true*, or for any node x in the tableau that is not a leaf, one of the following conditions hold:*

- **$\vee$-rule ($\vee$):** *There is a clause $\ell_1 \vee \ldots \vee \ell_n$ in $\mathcal{S}$ and the children of x are $\ell_1, \ldots \ell_n$.*
- **Superposition rule (S1):** *The constraints p_1 and p_2 are either x or ancestors of x and $p_1 \otimes p_2$ is not empty, and the children of x are the constraints s, for each pattern s in $p_1 \otimes p_2$. Otherwise, if $p_1 \otimes p_2$ is empty, x has the only child $FALSE$.*

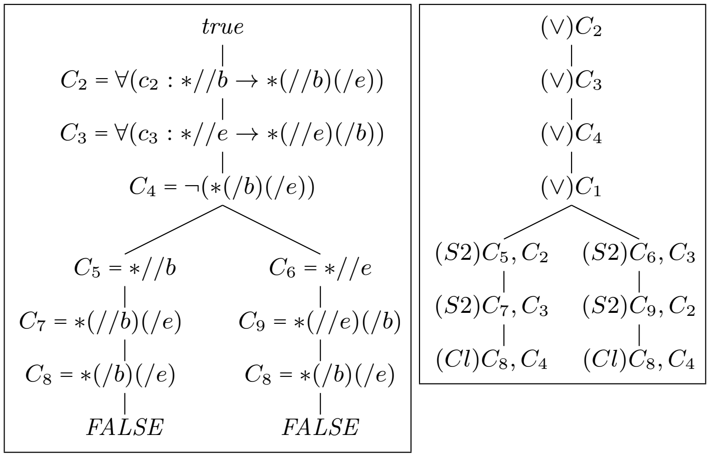

Fig. 5. A closed tableau for the specification S in Example 1 (left) and their applied rules (right)

- **Superposition rule (S2):** *The constraints p_1 and $\forall(c : p_2 \to q)$ are either x or ancestors of x such that there is a monomorphism $m : p_2 \to p_1$, and the children of x are the constraints s, for each s in $p_1 \otimes_{c,m} q$.*
- **Unfolding rule (U1):** *The constraint p where $p = p[n_1//hang(n_2)]$ is either x or an ancestor of x and the children of x are the constraint $p[n_1 \leftarrow /hang(n_2)]$, and the constraints $s[inc_2(n) \leftarrow //hang(n_2)]$ for each s in $p[n_1 \leftarrow] \otimes_{c,m} q$, where $q = prefix(n_1)[n_1 \leftarrow /n]$ with $label(n) = *$, and monomorphisms are depicted in the diagram below.*
- **Unfolding rule (U2):** *The constraint p where $p = p[n_1//hang(n_2)]$ is either x or an ancestor of x and the children of x are the constraint $p[n_1 \leftarrow /hang(n_2)]$, and the constraints $s[inc_2(n) \leftarrow /hang(n_2)]$ for each s in $p[n_1 \leftarrow] \otimes_{c,m} q$, where $q = prefix(n_1)[n_1 \leftarrow //n]$ with $label(n) = *$, and monomorphisms are depicted in the diagram below.*

$$\begin{tikzcd}
 & p[n_1 \leftarrow] \arrow[d, "inc_1"'] \\
prefix(n_1) \arrow[ur, "m"] \arrow[dr, "c"'] & s \\
 & q \arrow[u, "inc_2"]
\end{tikzcd}$$

- **Closing rule (Cl):** *The constraints p and $\neg q$ are either x or ancestors of x, such that there is a monomorphism $m : q \to p$, and x has the only child $FALSE$.*

Obviously the above rules not only describe if we can associate a given tableau to a specification $\mathcal{S}$, but they can also be used in the construction of a tableau for $\mathcal{S}$. The $\vee$-rule is the standard tableaux rule for creating the initial tableau from the clauses of the given specification. The superposition rules state that if we have two non-negative literals in a given branch, then we can extend that branch with the immediate consequences of these literals, as we have seen in Sec. 3.3. We may notice that the superposition rules define a finite number of children for a given node x, because the set of patterns resulting from a superposition of finite patterns is a finite set. The closing rule states that if, in a branch B, we have a positive and a negative literal, $p, \neg q$, that are contradictory, because p embeds q, then we can close B with $FALSE$.

Finally, the unfolding rules extend a branch with all the possible ways of unfolding an edge $n_1//n_2$. In principle, we considered two different ways of doing this unfolding: replacing $n_1//n_2$ by n_1/n_2 and $n_1/n//n_2$ (using rule U1) or by n_1/n_2 and $n_1//n/n_2$ (using rule U2), where $label(n) = *$ in both cases. However, it is necessary to take into account all possible identifications of such new node n with other nodes in the obtained literal. Otherwise, the rule may be unsound as explained in the following example.

Example 2. Consider the specification $\mathcal{S} = \{C_1, C_2, C_3, C_4\}$ with $C_1 = a(//b)(//c)$, $C_2 = \neg(a/b)$, $C_3 = \neg(a/c)$, and $C_4 = \neg(a(/*//b)(/*//c))$.
If rule U1 would just unfold an edge $//$ only in $/$ and $/*//$, then we could easily find a refutation for this specification as follows: By applying such U1 to the edge $a//b$ in C_1 we could get the constraints $C_5 = a(/b)(//c)$ and $C_6 = a(/*//b)(//c)$. The first literal, C_5, is directly contradictory with C_2. If we now unfold C_6 (on the edge $a//c$) by applying the rule U1, we could get the literals $C_7 = a(/*//b)(/c)$ and $C_8 = a(/*//b)(/*//c)$, which could be directly refuted with C_3 and C_4 respectively. But the specification $\mathcal{S}$ is not inconsistent because, for instance, the document $D = a/d(/b)(/c)$ satisfies all its clauses.
However, the situation with the defined unfolding rules is the following. The application of rule U1 on C_1 yields to the literals C_5, C_6, and the literal $C_9 = a/c//b$ obtained by joining the new node $*$ with the old node c in C_6; and the application of the rule U1 on C_6 yields to the literals C_7, C_8, and the literal $C_{10} = a/*(//b)(//c)$ obtained by joining the new node $*$ with the old node $*$ in C_8. While C_9 can be refuted with C_3, the literal C_{10} yields to an open branch as depicted in Fig. 6. See also Fig. 7 for the rules applied to built the tableau in Fig. 6.

Definition 11 (Open/closed branch, tableau proof). *In a tableau T a branch B is* closed *if B contains $FALSE$; otherwise, it is* open. *A tableau is closed if* all *its branches are closed. A* tableau proof *for (the unsatisfiability of) a specification $\mathcal{S}$ is a closed tableau T for $\mathcal{S}$ according to the rules given in Def. 10.*

Finally, it will be useful to define tableau satisfiability.

Definition 12 (Branch and tableau satisfiability). *A* branch *B in a tableau T is* satisfiable *if there exists a document $\mathcal{D}$ satisfying all the constraints in B. In this case, we say that $\mathcal{D}$ is a model for B, written $\mathcal{D} \models B$. A* tableau *T is* satisfiable *if there is a satisfiable branch B in T. If $\mathcal{D} \models B$ for a branch B in T, we also say that $\mathcal{D}$ is a model for T and also write $\mathcal{D} \models T$.*

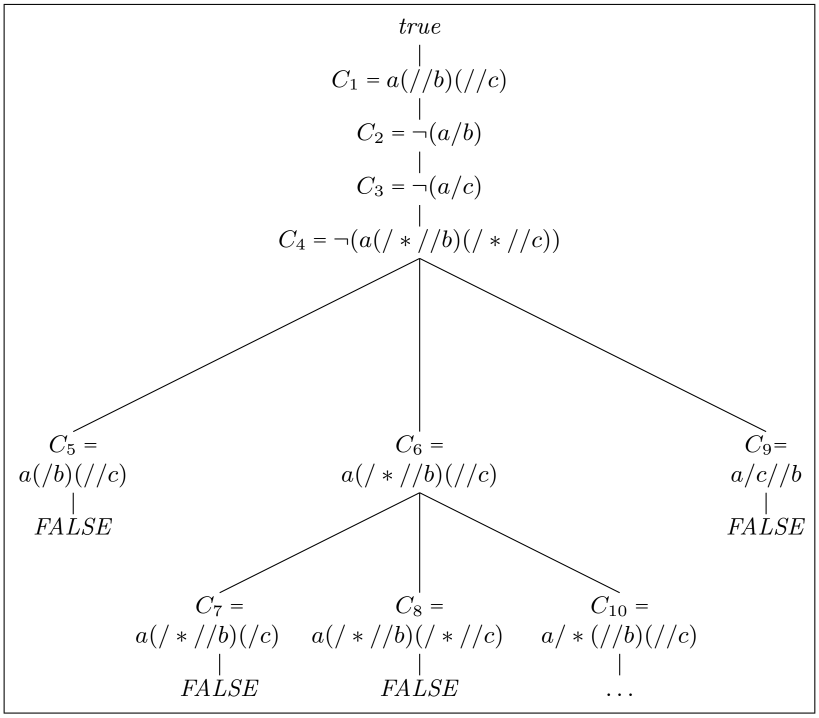

Fig. 6. An open tableau for the specification S in Example 2

5 Soundness and Completeness of the Tableau Method

In this section we prove that our tableau method is sound and complete. In particular, soundness means that if we are able to construct a tableau where all its branches are closed then our original specification $\mathcal{S}$ is unsatisfiable. Completeness means that if a *saturated* tableau includes an open branch, where the notion of saturation is defined below, then the original specification is satisfiable. Actually, the open branch provides a model that satisfies the specification.

Theorem 1 (Soundness). *If there is a tableau proof for the specification $\mathcal{S}$, then $\mathcal{S}$ is unsatisfiable.*

Proof. We prove that if a specification $\mathcal{S}$ is satisfiable, then any associated tableau cannot have all its branches closed. The proof is by induction on the structure of the tableau. Specifically, we show by induction on the construction of T that if $\mathcal{D} \models \mathcal{S}$ then $\mathcal{D} \models T$.

The base case is trivial since T consists only of the node $true$.

For the general case, assume that $\mathcal{D} \models T$ as inductive hypothesis. We have to show that if T' is constructed by applying a tableau rule to T, then $\mathcal{D} \models T'$. By induction, we know that there exists a branch B in T such that $\mathcal{D} \models B$. If this branch is not extended when constructing T', then it trivially holds that $D \models T'$ since B is still a branch in T'.

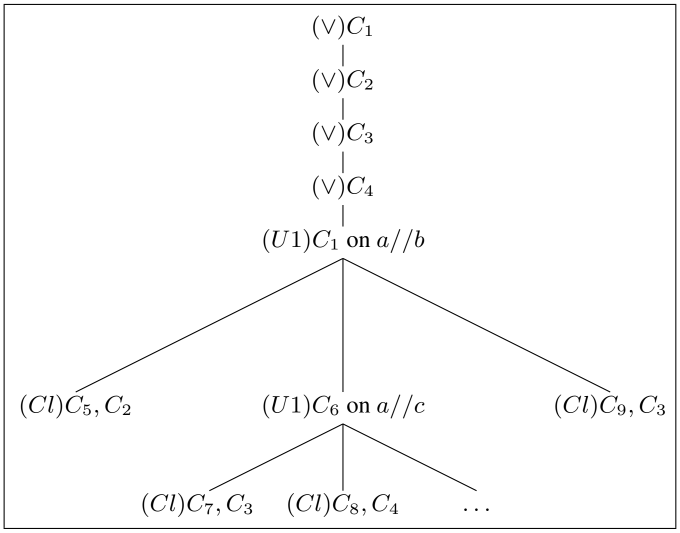

Fig. 7. The applied rules to build the tableau in Fig. 6

Otherwise, if B is extended, then we show that in T' there exists an extended branch B' from B such that $\mathcal{D} \models B'$ and therefore also $\mathcal{D} \models T'$. Now, we proceed by cases depending on what rule is applied in the extension:

- Suppose that the rule applied to construct T' is the $\vee$-rule. We know that one literal ℓ per clause in $\mathcal{S}$ exists such that $\mathcal{D} \models \ell$ because $\mathcal{D} \models \mathcal{S}$. The $\vee$-rule adds nodes labelled with literals from a clause in $\mathcal{S}$. Therefore, $\mathcal{D}$ must satisfy at least one of these literals.
- Suppose that the rule applied to construct T' is the superposition rule S1. Suppose p_1 and p_2 are the literals in B that are used for the extension. By inductive hypothesis we know that $\mathcal{D} \models p_1$ and $\mathcal{D} \models p_2$. It means that there are two monomorphisms $h_1 : p_1 \to \mathcal{D}$ and $h_2 : p_2 \to \mathcal{D}$. By Prop. 3, there exists some $s \in p_1 \otimes p_2$ verifying the *pair factorization property* with $h : s \to \mathcal{D}$ being a monomorphism, so, $\mathcal{D} \models s$.
- Suppose that the rule applied to construct T' is the superposition rule S2. Suppose p_1 and $\forall(c : p_2 \to q)$ are the literals in B that are used for the extension. By inductive hypothesis we know that $\mathcal{D} \models p_1$ and $\mathcal{D} \models \forall(c : p_2 \to q)$. Since $\mathcal{D} \models p_1$, there exists a monomorphism $h_1 : p_1 \to \mathcal{D}$. Then $h_1 \circ m$ is also a monomorphism from p_2 to $\mathcal{D}$. From here, since $\mathcal{D} \models \forall(c : p_2 \to q)$, there is a monomorphism $h_2 : q \to \mathcal{D}$ such that $h_1 \circ m = h_2 \circ c$. By Prop. 3, there exists some $s \in p_1 \otimes_{c,m} q$ verifying the *pair factorization property* with $h : s \to \mathcal{D}$ being a monomorphism,

so, $\mathcal{D} \models s$. Graphically:

$$\begin{tikzcd}
 & p_1 \arrow[d, "inc_1"'] \arrow[dr, "h_1"] & \\
p_2 \arrow[ur, "m"] \arrow[dr, "c"'] & s \arrow[r, "h"] & \mathcal{D} \\
 & q \arrow[u, "inc_2"] \arrow[ur, "h_2"'] &
\end{tikzcd}$$

- Suppose that the rule applied to construct T' is one of the unfolding rules. Suppose $p[n_1//hang(n_2)]$ is the literal in B that is used and that (U1) is the rule used for the extension. By inductive hypothesis we know that $\mathcal{D} \models p$. It means that there is a monomorphism $h_1 : p \to \mathcal{D}$, that is, it holds $h_1(n_1)//h_1(n_2)$. Then, accordingly to Cond. 5. in Def. 1 we have three cases:
  1. If $h_1(n_1)/h_1(n_2)$ holds, it is clear that there exists a monomorphism $h_2 : p[n_1 \leftarrow /hang(n_2)] \to \mathcal{D}$, so we have that $\mathcal{D} \models p[n_1 \leftarrow /hang(n_2)]$.
  2. If $h_1(n_1)/m_1//m_2/h_1(n_2)$ holds, for some nodes m_1 and m_2 in $\mathcal{D}$, then the following f_1 and f_2 are monomorphisms:
     - $f_1 : p[n_1 \leftarrow] \to \mathcal{D}$ such that $f_1(p[n_1 \leftarrow]) = h_1(p[n_1 \leftarrow])$,
     - $f_2 : q \to \mathcal{D}$ for $q = prefix(n_1)[n_1 \leftarrow /n]$ with $label(n) = *$, such that $f_2(prefix(n_1)) = h_1(prefix(n_1))$ and $f_2(n) = m_1$.

     Then, by Prop. 3, there exists some $s \in p[n_1 \leftarrow] \otimes_{c,m} q$ verifying the *pair factorization property* with $f : s \to \mathcal{D}$ being a monomorphism.

$$\begin{tikzcd}
 & p[n_1 \leftarrow] \arrow[d, "inc_1"'] \arrow[dr, "f_1"] & \\
prefix(n_1) \arrow[ur, "m"] \arrow[dr, "c"'] & s \arrow[r, "f"] & \mathcal{D} \\
 & q \arrow[u, "inc_2"] \arrow[ur, "f_2"'] &
\end{tikzcd}$$

     Therefore, $\mathcal{D} \models s[inc_2(n) \leftarrow //hang(n_2)]$ since we can define a monomorphism $h_2 : s[inc_2(n) \leftarrow //hang(n_2)] \to \mathcal{D}$ such that $h_2 = h_1$ except for:
     - $h_2(inc_2(n)) = f(inc_2(n)) = f_2(n) = m_1$
  3. Otherwise, $h_1(n_1)/m'/h_1(n_2)$ holds and it is enough to consider morphisms such that:
     - $f_2(n) = m'$,
     - $h_2(inc_2(n)) = f(inc_2(n)) = f_2(n) = m'$

  Similar arguments serve if the rule used is (U2).

Consequently, in all these cases, there exists an extended branch B' in T' such that $\mathcal{D} \models B'$ and therefore $\mathcal{D} \models T'$. ■

In order to prove completeness, the following notion of saturation of tableaux is required. Saturation describes some kind of fairness that ensures that we do not postpone indefinitely some inference step.

Definition 13 (Saturated Tableau). *Given a* tableau T *for a specification* $\mathcal{S}$*, we say that* T *is* saturated *if the following conditions hold:*

- *No new literals can be added to any branch* B *in* T *using the* $\vee$*-rule.*
- *For each branch* B *in* T*, one of the following conditions is satisfied:*
  - *either it is closed, or*
  - *it is open and all rules have been applied in* B*.*

It should be clear that it is always possible to build a (possibly infinite) saturated tableau. It is enough to keep, for every branch, a queue of the pending inferences.

To prove completeness we will show that we can associate a canonical model $\mathcal{D}_B$ to any open branch in a given tableau T so that, if T is saturated then $\mathcal{D}_B$ can be proven to be a model for T. In particular, this model is obtained by, first, computing r_B, which is the colimit of the diagram consisting of the patterns in the positive literals in the branch and the monomorphisms induced by rule applications. And, second, by replacing in r_B every $*$ label by a fresh label from Σ that is not present in any literal in the given specification. The existence of these colimits (satisfying an additional minimality property) is described in the Def. 14 and Prop. 4.

Definition 14 (Infinite colimits). *We say that* P *be a (possibly infinite)* directed diagram of patterns*, if it is a collection of patterns and monomorphisms between the patterns, such that for every pair of patterns* p_1 *and* p_2 *there exists a pattern* r *and monomorphisms* $f_1 : p_1 \to r$ *and* $f_2 : p_2 \to r$ *in* P*. We say that* P *has a* colimit *if there exists a pattern* r_P *together with a collection of morphisms* $\{h_p : p \to r_P \mid p \in P\}$ *such that, if* $f : p_1 \to p_2$ *in* P *then* $h_{p_1} = h_{p_2} \circ f$ *. Moreover, we say that the colimit is* minimal*, if for every finite pattern* q *such that there is a monomorphism* $g : q \to r_P$*, then, there is a pattern* p *in* P *and a monomorphism* $g_p : q \to p$ *such that the diagram below commutes:*

$$\begin{tikzcd}
q \arrow[rr, "g_p"] \arrow[dr, "g"'] & & p \arrow[dl, "h_p"] \\
& r_P &
\end{tikzcd}$$

Now, we show that every open branch defines a directed diagram, so that, if the tableau is saturated, it has a minimal colimit.

Proposition 4 (Colimit of open branches in saturated tableaux). *Given an open branch* B *in a saturated tableau* T *then, the set of patterns in positive literals in* B *is a directed diagram* P_B *that has a minimal colimit* r_B*.*

Proof. First of all, we define the diagram P_B associated to a branch B as follows:

- If p is a positive literal in a node of B then p is in P_B.

- If s, p_1 and p_2 are literals on nodes of B such that s is one of the child literals obtained from p_1 and p_2 after applying rule (S1), then the corresponding monomorphisms $inc_1 : p_1 \to s$ and $inc_2 : p_2 \to s$ are in P_B.
- If s, p_1 and $\forall(c : p_2 \to q)$ are literals on nodes of B, such that s is one of the child literals obtained from p_1 and $\forall(c : p_2 \to q)$ after applying rule (S2), then the corresponding monomorphism $inc_1 : p_1 \to s$ is in P_B.
- If s and p are literals on nodes of B such that s is one of the child literals obtained from p after applying rule (U1) or (U2), then the associated monomorphism $f : p \to s$ is in P_B.

Then, since T is saturated, if p and q are literals in B, the branch includes the application of the corresponding superposition rule (S1) to these literals. Moreover, since B is open, the superposition $p \otimes q$ is not empty and defines the morphisms $inc_1 : p_1 \to s$ and $inc_2 : p_2 \to s$, so P_B is a directed diagram.

For the colimit construction, it is enough to define r_B as the quotient of the union of the patterns in the diagram modulo the equivalence relation defined by the morphisms of the diagram[1]. For the minimality property, if q is a finite pattern such that $g : q \to r_P$, then there should exist a finite subset of patterns $P_0 \subseteq P_B$ such that the union of the nodes of the patterns in P_0 includes the set of nodes $g(n)$, where n is in $Nodes_q$. Then, using that P_B is directed we can prove the existence of a pattern p in P_B, obtained by doing superpositions on patterns in P_0, such that there is a monomorphism $g_p : q \to p$ verifying $g = h_p \circ g_p$. ■

Lemma 1 (Canonical models of saturated tableaux). *If B is an open branch of a saturated tableau T, r_B is its colimit, and $\mathcal{D}_B$ is the result of replacing in r_B each $*$ by a label a not present in the given specification $\mathcal{S}$, then $\mathcal{D}_B$ is a document such that $\mathcal{D}_B \models B$ and, hence, $\mathcal{D}_B \models \mathcal{S}$.*

Proof. First, we have to prove that $\mathcal{D}_B$ is indeed a document. This means proving that $\mathcal{D}_B$ satisfies Cond. 5 in Def. 1. Let n_1, n_2 be two nodes in $\mathcal{D}_B$ (and, hence, in r_B) such that $n_1//n_2$ holds. By Prop. 4, there must exist a pattern p in P_B containing such nodes n_1 and n_2 such that $n_1//n_2$ holds in p[2]. There are several possibilities:

1. If n_1/n_2 is in p then n_1/n_2 is in $\mathcal{D}_B$.
2. If $n_1//^d n_2$ is in p, since the tableau is saturated, at some point in the branch B we would have applied the unfolding rule (U1) to the literal p. As a consequence, B would also include the literal p_1 where either $p_1 = p[n_1 \leftarrow /hang(n_2)]$ or $p_1 = s[inc_2(n) \leftarrow //hang(n_2)]$ for one of the patterns s in $p[n_1 \leftarrow] \otimes_{c,m} q$, where $q = prefix(n_1)[n_1 \leftarrow /n]$ with $label(n) = *$. In the former case, we would know that in $\mathcal{D}_B$ we have n_1/n_2. In the latter case, B has the literal p_1 containing a node m_1 such that $n_1/m_1//^d n_2$ is in p_1. But because the tableau is saturated, at some later point in the branch B we would have applied the unfolding rule (U2) to the

[1] The least equivalence relation satisfying that if f is a morphism in the diagram and $f(n) = n'$, then $n \equiv n'$

[2] To be more precise there are nodes m_1, m_2, such that $m_1//m_2$ holds in p, $h_p(m_1) = n_1$ and $h_p(m_2) = n_2$.

literal p_1 on the edge $m_1//^d n_2$. As a consequence, B would also include the literal p_2 where either $p_2 = p_1[m_1 \leftarrow /hang(n_2)]$ or $p_2 = s'[inc_2(n') \leftarrow /hang(n_2)]$ for one of the patterns s' in $p_1[m_1 \leftarrow] \otimes_{c,m} q'$, where $q' = prefix(m_1)[m_1 \leftarrow //n']$ with $label(n') = *$. Now, in the former case, we would know that in $\mathcal{D}_B$ we have $n_1/m_1/n_2$. In the latter case, B has the literal p_2 containing a node m_2 such that $n_1/m_1//^d m_2/n_2$ is in p_2. Therefore we know that in $\mathcal{D}_B$ we have two nodes m_1 and m_2 such that $n_1/m_1//m_2/n_2$ holds.

3. Otherwise, there must be at least two edges between the nodes n_1 and n_2 in p. That is, there must be nodes m_1, m_2 in p with n_1/m_1 or $n_1//^d m_1$, and m_2/n_2 or $m_2//^d n_2$, such that $m_1 = m_2$ or $m_1//m_2$ holds in p. Now:
   (a) If n_1/m_1 and m_2/n_2 then trivially $n_1/m_1//m_2/n_2$ holds in $\mathcal{D}_B$.
   (b) If $n_1//^d m_1$ or $m_2//^d n_2$, then at some point we would apply the first or the second unfolding rule and, as in case 2, we would also prove that n_1, n_2 satisfy Cond. 5 in Def. 1.

Now, we prove that $\mathcal{D}_B$ satisfies each literal ℓ in B. Let $rename_B : r_B \to \mathcal{D}_B$ be the isomorphism that renames all the labels $*$ in $Nodes_{r_B}$ by a label a. Then, we have that $\mathcal{D}_B \models \ell$ if, and only if, $r_B \models \ell$ because, for every pattern p, the existence of a monomorphism $h : p \to \mathcal{D}_B$ implies the existence of a monomorphism $rename_B^{-1} \circ h : p \to r_B$; and, conversely, the existence of a monomorphism $f : p \to r_B$ implies the existence of a monomorphism $rename_B \circ f : p \to \mathcal{D}_B$. Therefore, it will be enough to prove that r_B satisfies each literal ℓ in B. We proceed by cases:

- If $\ell = p$ then, as a direct consequence of the colimit construction, we know that there exists a monomorphism $h_p : p \to r_B$, so $r_B \models p$.
- Assume that $\ell = \forall(c : p \to q)$. We will prove that, if there exists $h : p \to r_B$ then there is a monomorphism $f : q \to r_B$ such that $h = f \circ c$. First, we know as a consequence of the colimit construction that since p is finite, there is a pattern $r \in P_B$ (with the corresponding monomorphism $h_r : r \to r_B$) and a monomorphism $g_r : p \to r$, such that the following diagram commutes:

$$\begin{tikzcd}
p \arrow[rr, "g_r"] \arrow[dr, "h"'] & & r \arrow[dl, "h_r"] \\
& r_B &
\end{tikzcd}$$

Moreover, since T is saturated, we know that the superposing rule (S2) has been applied between $\forall(c : p \to q)$ and r. Suppose that $s \in r \otimes_{c,g_r} q$ is the one such that s is the literal that was hanged in the branch B. Then, by colimit definition, we know there is a monomorphism $h_s : s \to r_B$ and the following diagram defines the required monomorphism $f = h_s \circ inc_2 : q \to r_B$ such that $h = h_r \circ g_r = f \circ c$.

$$\begin{tikzcd}
& r \arrow[d, "inc_1"'] \arrow[dr, "h_r"] & \\
p \arrow[ur, "g_r"] \arrow[dr, "c"'] & s \arrow[r, "h_s"] & r_B \\
& q \arrow[u, "inc_2"] \arrow[ur, "f"'] &
\end{tikzcd}$$

– Let $\ell = \neg p$ and let us see that assuming that there exists $h : p \to r_B$ leads to a contradiction. If it was the case, we know that, since p is finite, there is a pattern $r \in P_B$ (with $h_r : r \to r_B$) and a monomorphism $g_r : p \to r$, such that the following diagram commutes:

$$\begin{tikzcd}
p \arrow[rr, "g_r"] \arrow[dr, "h"'] & & r \arrow[dl, "h_r"] \\
& r_B &
\end{tikzcd}$$

Then, since T is saturated the closing rule (C) must be applied between $\neg p$ and r so the branch should be closed, contradicting the premise. ■

Finally, we prove completeness of the tableau method.

Theorem 2 (Completeness). *If the specification $\mathcal{S}$ is unsatisfiable, then there is a tableau proof for $\mathcal{S}$.*

Proof. If there is no tableau proof for $\mathcal{S}$, then every tableau for $\mathcal{S}$ has an open branch. Hence, if T is a saturated tableau for $\mathcal{S}$, it should have an open branch B and, by Lemma 1, $\mathcal{D}_B \models \mathcal{S}$. ■

6 Related work

XPath [17, 21] is a well-known language for navigating an XML document (or XML tree) and returning a set of answer nodes. Since XPath is used in many XML query languages as XQuery, XSLT or XML Schema among others [20, 18, 19], a great amount of papers deal with different aspects on different fragments of XPath. For instance, in [4] an overview of formal results on XPath is presented concerning the expressiveness of several fragments, complexity bounds for evaluation of XPath queries, as well as static analysis of XPath queries. In [3] they study the problem of determining, given a query p (in a given XPath fragment) and a DTD D, whether there exists an XML document conforming to D and satisfying p. They show that the complexity ranges from PTIME to undecidable, depending on the XPath fragment and the DTD chosen. The work presented in [5] deals with the same problem (in a particular case) and it uses Hybrid Modal Logic to model the documents and some class of schemas and constraints. They provide a tableau proof technique for constraint satisfiability testing in the presence of schemas.

Our approach is different than the previous ones in two aspects. On the one hand, we do not consider any DTD or schema, and we use a simple fragment of XPath. In this sense our approach is simpler than previous ones. But, on the other hand, our aim is to define specifications of classes of XML documents as sets of constraints on these documents, and to provide a form of reasoning about these specifications. In this sense, our main question is satisfiability, that is, given a set of constraints S, whether there exists an XML document satisfying all constraints in S.

Some other work, which shares part of our aims, is the approach for the specification and verification of semi-structured documents based on extending a fragment of first-order logic [2, 12]. They present specification languages that allow us to specify classes of documents, and tools that allow us to check whether a given document (or a set of documents) follows a given specification. However, they do not consider the problem of defining deductive tools to analyze specifications, for instance to look for inconsistencies. Schematron [9] has a more practical nature. It is a language and a tool that is part of an ISO standard (DSDL: Document Schema Description Languages). The language allows us to specify constraints on XML documents by describing directly XML patterns (using XML) and expressing properties about these patterns. Then, the tool allows us to check if a given XML document satisfies these constraints. However, as in the previous approach, Schematron provides no deductive capabilities. Finally, the approach presented in this paper is very related to the work presented in [15, 16], showing how to use graph constraints as a specification formalism, and how to reason about these specifications. However, as discussed in Sec. 3, the descendant relation in our constraints makes non-trivial the application of the techniques in [15, 16]. In particular, the descendent relation would be second-order in the logic of graph constraints defined in [15, 16].

In [1, 13] we presented some preliminary work directly related to the work in this paper. In particular, in [13], we introduced the three kinds of constraints considered here and three main inference rules called R1, R2 and R3 (similar to the rules Cl, S1 and S2 in this paper). We proved that these rules were sound, but some counter-examples showed that they were not complete. So, we introduced two new rules, called Unfold1 and Unfold2 (a preliminary version of U1 and U2), that solved these counter-examples, so we conjectured that this was enough to prove completeness. Unfortunately, Unfold1 and Unfold2 are unsound as we explain in Example2. Moreover, we presented some subsumption and simplification rules in order to produce a more efficient procedure. In parallel, we implemented a prototype tool for reasoning with these rules that is described in [1] by means of examples and screenshots.

7 Conclusion and further work

In this paper, we have presented an approach for specifying the structure of XML documents using three kinds of constraints based on XPath, together with a sound and complete method for reasoning about them.

We strongly believe that satisfiability problem for this class of constraints is only semidecidable, since we believe that it would be similar to the (un)decidability of the satisfiability problem for the Horn clause fragment of first-order logic. As a consequence, if a given specification is inconsistent, we can be sure that our procedure will terminate showing that unsatisfiability. However, our procedure may not terminate if the given specification is satisfiable. In this context, we may consider that studying the complexity of a procedure that may not terminate is not very useful. Nevertheless, we may like to have an idea about the performance of our approach when the procedure terminates. One could think, that this performance would be quite poor, since checking if there is a monomorphism between two trees (a basic operation in our deduction procedure) is an NP-complete problem [10]. Actually, this is not our experience with the tool that we have implemented [1]. We think that the situation is similar to what happens with graph transformation tools. In these tools, applying a graph transformation rule means finding a subgraph isomorphism, which is also a well-known NP-complete problem. However, the fact that the graphs are typed (in our case, the trees are labelled), in practice, reduces considerably the search.

In the future, we plan to extend our approach to consider also cross-references and properties about the contents of documents. The former problem means, in fact, to extend our approach to graphs and graph patterns. For the latter case, we plan to follow the same approach that we used to extend our results for graphs in [15, 16] to the case of attributed graphs in [14].

References

1. Albors, J., and Navarro, M. *SpecSatisfiabilityTool: A tool for testing the satisfiability of specifications on XML documents*, Proceedings of PROLE 2014, EPTCS 173 (2015), 27-40.
2. Alpuente, M., Ballis, D., and Falaschi, M. *Automated Verification of Web Sites Using Partial Rewriting*, Software Tools for Technology Transfer, 8 (2006), 565-585.
3. Benedikt, M., Fan, W., and Geerts, F. *XPath satisfiability in the presence of DTDs*. JACM 55, 2 (2008).
4. Benedikt, M., and Koch, C. *XPath Leashed*, ACM Computing Surveys 41, 1 (2008).
5. Bidoit, N., and Colazzo D. *Testing XML constraint satisfiability*. Proceedings of the International Workshop on Hybrid Logic (HyLo 2006). ENTCS 174, 6 (2007), 45-61.
6. Habel A., Pennemann K.H. *Correctness of high-level transformation systems relative to nested conditions*, Mathematical Structures in Computer Science 19(2), (2009), 245–296.
7. Habel A., Radke H. *Expressiveness of graph conditions with variables*, International Colloquium on Graph and Model Transformation GraMoT 2010, ECEASST 30, (2010).
8. Hähnle, R. *Tableaux and Related Methods*, in Robinson,J.A.,Voronkov,A.(eds.) Handbook of Automated Reasoning (2001), 100–178.
9. Jelliffe, R. *Schematron*, Internet Document, http://xml.ascc.net/resource/ schematron/.
10. Kilpelainen, P., Mannila, H. *Ordered and Unordered Tree Inclusion*. SIAM Journal on Computing archive Volume 24 (2):340-356, (1995).
11. Miklau, G., and Suciu, D. *Containment and equivalence for a fragment of XPath*, JACM, 51, 1 (2004), 2-45.
12. Nentwich, C., Emmerich, W., Finkelstein, A., and Ellmer, E. *Flexible Consistency Checking*, ACM Transaction on Software Engineering and Methodology, 12(1) (2003), 28–63.
13. Navarro, M., and Orejas, F. *A refutation procedure for proving satisfiability of constraint specifications on XML documents*, SCSS 2014, EasyChair EPiC series, 30 (2014), 47-61.
14. Orejas, F. *Symbolic Graphs for Attributed Graph Constraints*, Journal of Symbolic Computation. 46(3) (2011), 294–315.
15. Orejas, F., Ehrig, H., and Prange, U. *A Logic of Graph Constraints*, Fundamental Approaches to Software Engineering, 11th Int. Conference, FASE 2008. LNCS 4961 (2008) 179-198.
16. Orejas, F., Ehrig, H., and Prange, U. *Reasoning with graph constraints*, Formal Asp. Comput. 22, 3-4 (2010), 385-422.
17. WORLD WIDE WEB CONSORTIUM. 1999a. *XML path language (XPath) recommendation*, http://www.w3c.org/TR/XPath/.
18. WORLD WIDE WEB CONSORTIUM. 1999b. *XSL transformations (XSLT). W3C recommendation version 1.0*, http://www.w3.org/TR/xslt.

19. WORLD WIDE WEB CONSORTIUM. 2001. *XML schema part 0: Primer. W3C recommendation*, http://www.w3c.org/XML/Schema.
20. WORLD WIDE WEB CONSORTIUM. 2002. *XQuery 1.0 and XPath 2.0 formal semantics. W3C working draft*, http://www.w3.org/TR/query-algebra/.
21. WORLD WIDE WEB CONSORTIUM. 2007. *XML path language (XPath) 2.0.*