

Capturing well typed references in DTDs

Nicole Bidoit and Dario Colazzo
LRI, UMR CNRS 8623,
Université Paris 11, Bat 490
91405 Orsay, France
{nicole.bidoit,dario.colazzo@lri.fr}

July 23, 2006

Abstract

Surprisingly enough, there has been few investigations for typing references of semistructured data and XML documents. This paper build on a previous proposal [7] introducing simple schemas with well-typed references and showing that such schemas, called normalized ref-schemas, are expressible as formulas of Hybrid Modal Logic. The aim of the present paper is to extend normalized ref-schemas in order to allow one for general regular expressions and provide a fully general notion of schema capturing well-typed references, called ref-schemas. The main contribution of the paper is to show that ref-schemas are still expressible in Hybrid Modal Logic which entails that tools like for instance the tableau system developed in [9] can be used in order to check for constraint satisfiability in presence of ref-schemas.

1 Introduction

Having a well defined notion of schema for semistructured data and XML documents is widely recognized as a key feature for query optimization, update manipulation and automated reasoning over such data in general. Since semistructured data and XML docu-

ments contain references, an adequate notion of schema should provide a mechanism for specifying the “types” of the referenced elements. Surprisingly enough, this is either not provided by the notions of schema currently proposed in the literature or it is provided indirectly [27, 26]. We would like to stress that problems like subtyping, constraint implication and satisfiability, query correctness etc become trivial to state and easier to investigate when the same formalism (logic as a matter of fact) is used to describe schemas, constraints and queries. Indeed, the main purpose of this paper is to show that *Hybrid Modal Logic* (HML)[11] is a promising formalism for specifying and reasoning about sophisticated schemas, constraints and queries.

Indeed, HML was previously investigated by [1, 20] for expressing constraints. The work in [7] shows that HML is powerfull enough to express, of course, quite general constraints subsuming path constraints [15]. In [7], a rather simple proposal of schema capturing well-typed references (in the unordered setting) is made. In a way, these schemas further called normalized ref-schemas extend *normalized DTDs* [5]. The logical formalization of such schemas in terms of HML formulas is investigated and leads to show that: a document conforms to a schema \mathcal{G} iff it satisfies a HML

formula $\tau_{\mathcal{G}}$. A slight extension of HML needs to be done in order to define queries and leads to an easy translation of Xpath queries.

In [9], thanks to the formalization of schemas as HML formulas, we naturally address constraint satisfiability in presence of schema: given a schema \mathcal{G} and a constraint \mathcal{C} , does there exist a document conforming to \mathcal{G} and satisfying \mathcal{C} ? This question reduces to: is $\tau_{\mathcal{G}} \wedge \mathcal{C}$ finitely satisfiable? [9] provides a proof procedure based on tableau techniques for testing satisfiability by generating models in the case of non recursive schemas.

Because the notion of normalized ref-schemas introduced in [7] does not allow one for general regular expressions, in a first step, the present paper provides a fully general notion of schema capturing well-typed references, further called ref-schemas. More importantly, we show that this generalization is still expressible in HML. This is done as follows: given a ref-schema \mathcal{G} we provide a reduction of \mathcal{G} to a normalized ref-schema \mathcal{G}_n together with a constraint $\mathcal{C}_{\mathcal{G}}$ expressed in HML meaning that, given an instance \mathfrak{M} , there exists an instance \mathfrak{M}_n of \mathcal{G}_n such that \mathfrak{M} is conforming to \mathcal{G} iff \mathfrak{M}_n is conforming to \mathcal{G}_n and satisfies the constraint $\mathcal{C}_{\mathcal{G}}$. This translation from ref-schemas to normalized ref-schemas implies, for instance, that the tableau system presented in [9] can be used for constraint satisfiability checking in presence of ref-schemas. It also opens new directions for investigating other problems linked to query optimization.

The paper is organized as follows. The next section is devoted to introducing Hybrid Modal Logic and to showing its relationship with semistructured data and XML. Section 3 first defines general schemas capturing references as “first class citizen” and then recall the definition of normalized ref-schema¹ previ-

ously introduced in [7] although it is slightly extended here. Of course, Section 3 provides the characterization of a document conforming to a normalized ref-schema in terms of a model satisfying a set of HML formulae. Section 4 is devoted to showing how general ref-schema can be expressed in HML by mapping such a schema to a normalized one together with structural constraints expressed in HML. Section 5 is a short overview² of a tableau proof system for checking constraint satisfiability in presence of non recursive ref-schema. Related work and further research directions are discussed in Section 6 and Section 7.

2 Hybrid Modal Logic

We assume the reader familiar with modal logics and just recall here the main features of *Hybrid modal logic* (HML) [13, 2, 3]. Previous work [1, 20, 17, 7] have investigated a modal logic approach to modelize semistructured data which is naturally motivated by the fact that such data are commonly viewed as edge labelled graphs thus as Kripke models [25]. Although modal logic is a simple formalism for working with graphs, it has no mechanism for referring to and reasoning about the individual nodes in such structures. HML increases the effectiveness of modal logic by allowing one to grasp the nodes via formulas: it provides a mechanism to name states (or graph nodes) and to assert that a formula is true at a named state (or graph node). This is made possible by four fundamental features: (1) a *nominal* or a *state variable* is a special atomic formula that names or denotes the unique state where the formula holds; (2) the *satisfaction operator* $@_u$ applied to a formula ψ enables to check satisfaction of ψ at the state named (or denoted) by the nominal (or state variable) u ; (3) the *binder operator* $\downarrow x$ applied to a formula

¹In [7], normalized ref-schemas are called pattern grammars.

²The full presentation can be found in [9].

ψ binds the state variable x in ψ to the current state.

The *alphabet* of a HML language is given by four disjoint sets: propositions $PROP = \{p, q, \dots\}$, nominals $NOM = \{a, b, \dots\}$, state variables $SVAR = \{x, y, \dots\}$ and labels $\mathcal{E} = \{e_1, \dots, e_n\}$. *Well formed formulas* (wffs) are defined by:

WFF ::= p | \top | $\neg\psi$ | $\psi_1 \wedge \psi_2$ | $[e]\psi$ | u | $\downarrow x \psi$ | $@_u\psi$

where ψ , ψ_1 and ψ_2 are wffs, $p \in PROP$, $x \in SVAR$ and $u \in NOM \cup SVAR$.

The operators \vee , \rightarrow and $\langle e \rangle$ are defined by:

$\psi_1 \vee \psi_2 =_{def} \neg(\neg\psi_1 \wedge \neg\psi_2)$,
 $\psi_1 \rightarrow \psi_2 =_{def} \neg\psi_1 \vee \psi_2$ and
 $\langle e \rangle\psi =_{def} \neg[e]\neg\psi$.

A *model* (a document) \mathfrak{M} of HML is a Kripke structure $(S, r, R, V, \mathcal{I}_{nom})$ where: S is a finite set of *states* (the nodes of the document) containing a distinguished element r (the root of the document); $R = \{r_e | e \in \mathcal{E}\}$ is a set of binary *accessibility relations* on S (the labelled edges of the document linking nodes); the function $V: PROP \rightarrow Pow(S)$ assigns to each proposition p the set of states where p holds (the data component of the document); the function $\mathcal{I}_{nom}: NOM \rightarrow S$ assigns a unique state to each nominal (see example below).

A valuation $g : SVAR \rightarrow S$ assigns a state to each state variable. By $g \stackrel{x}{\sim} g'$ we denote that g' is a x -variant of g .

The presentation of the semantics of HML is restricted to hybrid features. A model \mathfrak{M} satisfies the wff ψ at state s wrt a valuation g , noted $\mathfrak{M}, g, s \models \psi$, if:

- $\mathfrak{M}, g, s \models p$ iff $s \in V(p)$, $p \in PROP$
- $\mathfrak{M}, g, s \models a$ iff $\mathcal{I}_{nom}(a) = s$
- $\mathfrak{M}, g, s \models x$ iff $g(x) = s$
- $\mathfrak{M}, g, s \models \psi_1 \wedge \psi_2$ iff $\mathfrak{M}, g, s \models \psi_1$ and $\mathfrak{M}, g, s \models \psi_2$
- $\mathfrak{M}, g, s \models \neg\psi$ iff $\mathfrak{M}, g, s \not\models \psi$

- $\mathfrak{M}, g, s \models [e]\psi$ iff for any $s' \in S$, $(s, s') \in r_e$ implies $\mathfrak{M}, g, s' \models \psi$
- $\mathfrak{M}, g, s \models \downarrow x \psi$ iff $\mathfrak{M}, g', s \models \psi$ with $g \stackrel{x}{\sim} g'$, $g'(x) = s$
- $\mathfrak{M}, g, s \models @_x\psi$ iff $\mathfrak{M}, g, g(x) \models \psi$
- $\mathfrak{M}, g, s \models @_a\psi$ iff $\mathfrak{M}, g, \mathcal{I}_{nom}(a) \models \psi$

The language is extended with two dual modalities G and F where $\mathfrak{M}, g, s \models G\psi$ iff for any state s accessible via a path from the current state, ψ holds at s and $F\psi =_{def} \neg G\neg\psi$. We also use $G^*\psi$ for $\psi \wedge G\psi$ and $F^*\psi$ for $\psi \vee F\psi$. Recall that when such modalities are added, HML is no longer a fragment of first order logic.

The next example illustrates how a semistructured data is mapped to a model of an HML language and illustrates the semantics of the hybrid operators.

In the following, when a node in a model \mathfrak{M} is reached by an \tilde{e} edge, the node is said to be an \tilde{e} node.

Example 1 *Figure 1 gives an example of a document that is of a Kripke model \mathfrak{M} . It is assumed that the only nominal of our language is root which is intuitively used to name the root of a document and hence that $\mathcal{I}_{nom}(root) = r$. This assumption is kept in the rest of the paper. Note that r in the figure is placed over the root of the document and is not a proposition. Proposition are placed inside the nodes of the document. Note also that, in this example, no proposition is assigned to internal nodes. Finally, next, a node which is the target of a \tilde{e} label is called a \tilde{e} node.*

Now let us use the example to illustrate the semantics of HML formulas. The property which says that the edges outgoing from the root of the document are only e edges is expressed by: $@_{root}(\neg\langle p \rangle \top \wedge \neg\langle o \rangle \top \wedge \neg\langle \vec{r} \rangle) \top$.

The property which says that an e node should be the source of exactly one reference \vec{r}

if it is not a leaf node is expressed by : $G^*\langle e \rangle \downarrow x (F\top \rightarrow (@_x \langle \vec{r} \rangle \downarrow y @_x [\vec{r}] y))$.

These two properties are satisfied by the document depicted in Figure 1 or in other word $\mathfrak{M}, g, r \models \varphi$ holds for any valuation g and for φ being one of the two above HML wffs.

3 Ref-Schemas : well-typed references

This section is devoted to the presentation of a general notion of schema *a la* DTD capturing well typed references.

Next, we assume that \mathcal{V} is a finite set of non-terminal symbols, containing the symbol *Start*, and the empty word Λ . By convention, a non terminal symbol starts with a capital letter while a label starts with a non-capital letter. The set of labels \mathcal{E} is partitionned in two disjoint sets E and \vec{E} : labels in E are called *child* labels whereas labels in \vec{E} are called *references*. We use the following convention : e (resp. \vec{e} , \tilde{e}) denotes a child label (resp. a reference, any label). For the sake of the presentation, we avoid to consider base types.

Definition 1 (ref-schema) A ref-schema \mathcal{G} is given by $(\mathcal{E}, \mathcal{V}, Start, \theta)$ where $\mathcal{E} = E \cup \vec{E}$, \mathcal{V} and *Start* are defined as above and where the typing function θ associates to each non terminal symbol X a regular expression of the form: $R ::= \tilde{e}X \mid R + R \mid R, R \mid R^* \mid \Lambda$.

The typing function θ needs to satisfy that:

- (1) for each child-label e , $type(e)$ is a singleton where $type(\tilde{e})$ denotes the set of non terminals occuring in elementary pattern of the form $\tilde{e}X$ in \mathcal{G} , and
- (2) for each non terminal X distinct from *Start*, $Start \Rightarrow_{\mathcal{G}}^* X$ holds where $\Rightarrow_{\mathcal{G}}^*$ is the transitive closure of the relation $\Rightarrow_{\mathcal{G}}$ naturally defined by $X \Rightarrow_{\mathcal{G}} Y$ if Y occurs in $\theta(X)$.

Note that condition (1) over typing function is the usual one considered for DTD. Intuitively,

condition (2) ensures that all (definitions of) types are linked to the initial type *Start*.

In order to define what are the documents conforming to a ref-schema, \mathcal{G} we need to associate to a regular expression R its extension $\llbracket R \rrbracket$. The multi-set $\llbracket R \rrbracket$ is defined according to the following equations, where \uplus denotes union over multi-sets:

$$\begin{aligned} \llbracket \tilde{e}X \rrbracket &= \{\{\tilde{e}X\}\} \\ \llbracket R_1 + R_2 \rrbracket &= \llbracket R_1 \rrbracket \cup \llbracket R_2 \rrbracket \\ \llbracket R_1, R_2 \rrbracket &= \{\{b_1 \uplus b_2 \mid b_i \in \llbracket R_i \rrbracket\}\} \\ \llbracket R^* \rrbracket &= \{\{b_1 \uplus \dots \uplus b_n \mid b_i \in \llbracket R \rrbracket, n > 0\}\} \\ &\quad \cup \{\{\}\} \\ \llbracket \Lambda \rrbracket &= \{\{\}\} \end{aligned}$$

We are now ready to formally define what are the instances of a ref-schema. Recall here that for us a document is a (Kripke) model.

Definition 2 Let $\mathfrak{M} = (S, r, R, V, \mathcal{I}_{nom})$ be a finite model and $\mathcal{G} = (\mathcal{E}, \mathcal{V}, Start, \theta)$ be a ref-schema. We say that \mathfrak{M} satisfies the ref-schema \mathcal{G} , denoted $\mathfrak{M} : \mathcal{G}$, if:

- (1) the model \mathfrak{M} restricted to the relations r_e such that e is a child label, is a tree, and
- (2) there exists a total mapping $\vartheta : S \rightarrow \mathcal{V}$ such that:

- (a) $\vartheta(r) = Start$,
- (b) for all $n \in S$ if $\vartheta(n) = X$ and $\theta(X) = R$ then $\{\{\tilde{e}Y \mid (n, n') \in R_{\tilde{e}} \text{ and } Y = \vartheta(n')\}\} \in \llbracket R \rrbracket$

Example 2 The recursive schema \mathcal{G} specified below aims at describing trees whose root has an arbitrary number of e -nodes; each e -node can have an arbitrary number of either p -nodes or o -nodes as children; each p -node can have an even number of e -nodes as children; an o -node can have an odd number of e -nodes as children. Moreover, each e -node can have an arbitrary number of \vec{r} references pointing either to an o -node or a p -node.

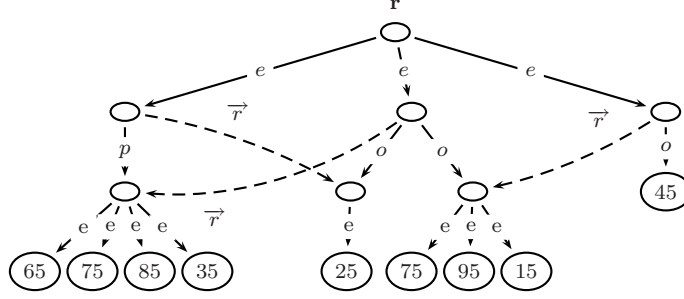


Figure 1: A document

$$\theta_0(\text{Start}) = (e X)^*,$$

$$\theta_0(X) = (p Y + o Z)^*, (\vec{r} Y + \vec{r} Z)^*$$

$$\theta_0(Y) = (e X, e X)^*$$

$$\theta_0(Z) = (e X, e X)^*, e X$$

Note that the document \mathfrak{M} of Figure 1 is an instance of this ref-schema i.e. $\mathfrak{M} : \mathcal{G}$.

For the sake of the translation of ref-schemas into normalized ones, we proceed to a simplification of the regular expressions considered. The reader should pay attention to the fact that this simplification is only possible because we are working under the unordered assumption.

Lemma 1 *Let R be a regular expression.*

There exists a simplified expression $\text{Simp}(R)$ such that $\llbracket R \rrbracket = \llbracket \text{Simp}(R) \rrbracket$ where a simplified expression is defined according to the following grammar:

$$R ::= B \mid (R + R)$$

$$B ::= \tilde{e}X \mid B, B \mid A^* \mid \Lambda$$

$$A ::= \tilde{e}X \mid A, A$$

Proof: We give below the rules that are to be used in order to construct the simplified expression $\text{Simp}(R)$ associated with R .

$$\text{Simp}(\tilde{e}X) = \tilde{e}X$$

$$\text{Simp}(R_1 + R_2) = \text{Simp}(R_1) + \text{Simp}(R_2)$$

$$\text{Simp}(R_1, R_2) = \text{Simp}(R_2, R_1)$$

$$\text{Simp}(R_1, R_2) = \text{Simp}(\text{Simp}(R_1), \text{Simp}(R_2))$$

$$\text{if } \exists i = 1, 2 \text{ } \text{Simp}(R_i) \text{ is a + expression}$$

$$\text{Simp}(R_1, R_2) = \text{Simp}(R_1), \text{Simp}(R_2)$$

$$\text{if } \forall i = 1, 2 \text{ } \text{Simp}(R_i) \text{ is not a + expression}$$

$$\text{Simp}((R_1 + R_2), R_3) = \text{Simp}(R_1, R_3) + \text{Simp}(R_2, R_3)$$

$$\text{Simp}((R_1 + R_2)^*) = \text{Simp}((R_1^*, R_2^*))$$

$$\text{Simp}((R_1^*, R_2^*)^*) = \text{Simp}(R_1, R_2, R_1^*, R_2^*) + \text{Simp}(R_2^*)$$

$$\text{Simp}((R_1^*)^*) = \text{Simp}((R_1^*))$$

$$\text{Simp}(\Lambda) = \Lambda$$

□

Example 3 *The simplification of the ref-schema given in example 2 yields a new typing function θ_1 defined as follows:*

$$\theta_1(\text{Start}) = (e X)^*,$$

$$\theta_1(X) = (p Y)^*, (o Z)^*, (\vec{r} Y)^*, (\vec{r} Z)^*$$

$$\theta_1(Y) = (e X, e X)^*$$

$$\theta_1(Z) = (e X, e X)^*, e X$$

Next, wlog, all expressions considered are simplified ones. A last slight modification of expressions is needed in order to cope with the following situation: in a ref-schema \mathcal{G} , it may happen that an elementary pattern $(\tilde{e}X)$ has multiple occurrences. Rather than disallowing such cases and in order to keep our framework fully general, we introduce markers in order to distinguish these occurrences. Markers are propositions. They are introduced in the definition of simplified regular expression by re-

placing $\tilde{e}X$ by $(\tilde{e}, p)X$ where p is a proposition symbol of the language.

Definitions 1 and 2 of a ref-schema and of ref-schema instances need to be slightly modified. The changes are technical but not very deep. Thus:

- A marked ref-schema \mathcal{G} is given by $(\mathcal{E}, \mathcal{V}, Start, \theta)$ where θ associates to each non terminal symbol X a marked simplified regular expression with conditions (1) and (2) of Definition 1 plus the following one:
(3) if $(\tilde{e}, p)X$ and $(\tilde{f}, q)Y$ occurs in \mathcal{G} then $p \neq q$.
- The definition of the extension $\llbracket R \rrbracket$ of R is updated by replacing $\llbracket \tilde{e}X \rrbracket = \{\{\{\tilde{e}X\}\}\}$ by $\llbracket (\tilde{e}, p)X \rrbracket = \{\{\{(\tilde{e}, p)X\}\}\}$.
- Finally, definition 2 is changed by rewriting condition (b) as: forall $n \in S$ if $\vartheta(n) = X$ and $\theta(X) = R$ then $\{\{(\tilde{e}, p)Y \mid (n, n') \in R_{\tilde{e}}, Y = \vartheta(n') \text{ and } p \in V(n')\}\} \in \llbracket R \rrbracket$.

Intuitively, if we consider a finite state automata associated to the schema, markers p can be seen as unique identifiers of automata states (assume that each $\tilde{e}X$ corresponds to an automata state). Then, over schema instances, the set of p 's associated to each node represent the set of automata states that successfully recognised that node (a node can be anaized in multiple states due to references). So proposition symbols associated to the instance nodes can actually be seen as a successful run of the schema automata.

For the sake of simplicity, in the following, we will omit the marker p for expressions $\tilde{e}X$ that only occur once.

Example 4 For our running example, after adding markers, we obtain the following typing function θ_2 :

$$\begin{aligned}\theta_2(Start) &= ((e, p_0)X)*, \\ \theta_2(X) &= (p Y)*, (o Z)*, (\vec{r} Y)*, (\vec{r} Z)* \\ \theta_2(Y) &= ((e, p_1)X, (e, p_2)X)* \\ \theta_2(Z) &= ((e, p_3)X, (e, p_4)X)*, (e, p_5)X\end{aligned}$$

[7] introduces schemas capturing well-typed references in a limited way. Intuitively, these schemas, further called normalized ref-schemas³ do not allow one for fully general expression and can be viewed as an extension of the normalized DTD of [5] with well typed references. The main contribution of [7] is to show that documents conforming to a given normalized ref-schemas \mathcal{G} are exactly the finite Kripke models satisfying a HML wff $\tau_{\mathcal{G}}$. This entails advantageously that a schema can be processed as any and together with other constraints.

Definition 3 A *normalized ref-schema* $(\mathcal{E}, \mathcal{V}, Start, \theta)$ is a (marked) ref-schema based on the normalized regular expressions defined by:

$$\begin{aligned}R &:= B \mid R + R, \text{ and} \\ B &:= \Lambda \mid ((\tilde{e}, p)X)^{op} \mid B, B \text{ where } op \text{ is either } ! \text{ or } *.\end{aligned}$$

For instance, normalized ref-schema forbid expression of the following form: $((\tilde{e}_1, p_1)X_1, \dots, (\tilde{e}_k, p_k)X_k)^*$.

The result presented below was stated and proved in [7] although without markers. Recall that when e is a child label, $type(e)$ is a singleton $\{X\}$ and thus we abusively write $type(e)$ instead of X . Next, for a reference \vec{e} , $child(\vec{e})$ denotes the set of child labels $\{e \mid e \in E \text{ and } type(e) \in type(\vec{e})\}$. For instance, with our running example, $type(e) = X$ and $child(\vec{r}) = \{p, o\}$ because $type(\vec{r}) = \{Y, Z\}$, $type(p) = Y$ and $type(o) = Z$.

³In [7], they are called pattern schemas.

In order to state this result we need to associate to each expression R an HML wff $\Psi(R)$ defined as follows:

1. If R is Λ then $\Psi(R) = \bigwedge_{\tilde{e} \in \mathcal{E}} \neg \langle \tilde{e} \rangle \top$
2. If R is of the form $R_1 + R_2$ then

$$\Psi(R) = \Psi(R_1) \vee \Psi(R_2)$$
3. If R is $((\tilde{e}_1, p_1)X_1)^{op_1}, \dots, ((\tilde{e}_k, p_k)X_k)^{op_k}$ then

$$\Psi(R) = \bigwedge_{i=1 \dots k} \tau_i \wedge \bigwedge_{e \text{ not in } \theta(X)} \neg \langle e \rangle \top$$

if $op_i = !$ then

$$\tau_i = \downarrow x \langle \tilde{e}_i \rangle \downarrow y (p_i \wedge @_x[\tilde{e}_i](p_i \rightarrow y))$$

if $op_i = *$ then

$$\tau_i = \langle \tilde{e}_i \rangle (\top \rightarrow \bigvee_{p \in Prop_{\tilde{e}_i}} p)$$

with $Prop_{\tilde{e}_i} = \{p \mid ((\tilde{e}_i, p)Y)^{op} \text{ in } \theta(X)\}$

Theorem 1 [7] *Let \mathcal{G} be a normalized ref-schema and let \mathfrak{M} be a model (document). We have:*

$$\mathfrak{M} : \mathcal{G} \text{ iff } \mathfrak{M}, g, r \models \tau_{\mathcal{G}}$$

where g is any state variable valuation and $\tau_{\mathcal{G}}$ is the HML wff tree $\wedge \tau_{\mathcal{G}}^E \wedge \tau_{\mathcal{G}}^{\vec{E}}$ with:

- tree is the wff ensuring that the “subframe” of \mathfrak{M} generated by child labels is a tree⁴

- $\tau_{\mathcal{G}}^E$ is $@_{root}(\tau_{Start} \wedge \bigwedge_{e \in E} G^*[e]\tau_{type(e)})$ where $\tau_{type(e)} = \Psi(\theta(type(e)))$.

- $\tau_{\mathcal{G}}^{\vec{E}}$ is $@_{root}(\bigwedge_{\vec{e} \in \vec{E}} G^*[\vec{e}]\downarrow x (\bigvee_{e \in child(\vec{e})} @_{root} F^*\langle e \rangle x))$

Intuitively, the wff $\tau_{\mathcal{G}}^E$ essentially checks that, given a state x reachable by an e child edge, the outgoing edges (child edges as well as references) are the ones allowed by the schema. The wff $\tau_{\mathcal{G}}^{\vec{E}}$ is concerned with reference type checking: it checks the type of the node which are the target of references.

⁴For sake of simplicity, the wff *tree* is not presented here.

4 How to express ref-schema in HML

This section is devoted to show that HML is powerful enough to express ref-schemas. Intuitively, we show that a ref-schema is equivalent to a normalized ref-schema together with a constraint. Because the constraint and the normalized ref-schema are HML formulas, this entails that general ref-schema are expressible in HML. We would like to stress that our motivation to proceed to such a translation via normalized ref-schema (rather than to a direct one) is the tableau based technique that we have developed in [9] for checking constraint satisfiability in presence of normalized ref-schema. Hence our translation not only serves to show that general ref-schemas are expressible in HML but also leads immediately to make our tableau system available for checking constraint satisfiability in presence of general ref-schema.

The next result formalizes the fact that general ref-schemas are expressible in HML.

Theorem 2 *Let \mathcal{G} be a marked ref-schema. Then there exists a normalized ref-schema \mathcal{G}_{norm} and an HML constraint $\mathcal{C}_{\mathcal{G}}$ such that:*

1. for each model \mathfrak{M} there exists a model \mathfrak{M}_{norm} such that $\mathfrak{M} : \mathcal{G}$ iff $\mathfrak{M}_{norm} : \mathcal{G}_{norm}$ and $\mathfrak{M}_{norm}, g, r \models \mathcal{C}_{\mathcal{G}}$.
2. for each model \mathfrak{M}_{norm} there exists a model \mathfrak{M} such that $\mathfrak{M} : \mathcal{G}$ iff $\mathfrak{M}_{norm} : \mathcal{G}_{norm}$ and $\mathfrak{M}_{norm}, g, r \models \mathcal{C}_{\mathcal{G}}$.

Before sketching the proof of Theorem 2, let us highlight that: on the one hand, because \mathcal{G}_{norm} is a normalized ref-schema, Theorem 1 implies that HML can express general ref-schemas; on the other hand, because of item 2 of theorem 2, given a constraint \mathcal{C} , checking satisfiability of \mathcal{C} in the presence of the schema \mathcal{G} reduces to checking satisfiability of $\mathcal{C} \wedge \mathcal{C}_{\mathcal{G}}$ in the presence of \mathcal{G}_{norm} .

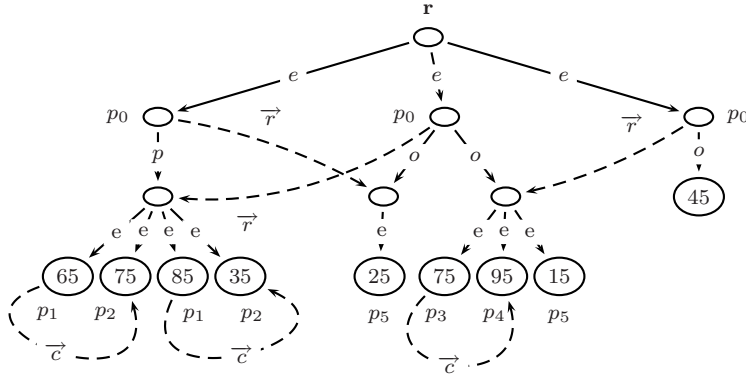


Figure 2: A document

Sketch of proof : We proceed in a constructive manner to exhibit the normalized schema \mathcal{G}_{norm} and the constraint $\mathcal{C}_{\mathcal{G}}$. Let $\mathcal{G} = (\mathcal{E}, \mathcal{V}, Start, \theta)$ be marked ref-schema. We associate to \mathcal{G} the normalized ref-schema $Norm(\mathcal{G}) = (\mathcal{E}', \mathcal{V}, Start, \theta')$ where:

- $\mathcal{E}' = \mathcal{E} \cup \{\vec{c}\}$ where \vec{c} is a new reference,
- θ' is obtained in two steps. Each maximal sub-expression of the form $((\tilde{e}_1, p_1)X_1, \dots, (\tilde{e}_k, p_k)X_k)^*$ is first replaced by $((\tilde{e}_1, p_1)X_1)^*, \dots, ((\tilde{e}_k, p_k)X_k)^*$. And each remaining sub-expression (\tilde{e}, p) is replaced by $(\tilde{e}, p)X!$

Then for $i=1$ to $k-1$, $\theta(X_i)$ (which already may have been modified by the previous action) is replaced by the simplified expression $Simp(\theta(X_i), (\vec{c} X_{i+1})^*)$. Note that we do not associate any proposition with the new reference \vec{c} .

For instance, when applied to the simplified and marked version of our running example, this transformation yields the new typing function θ_3 :

$$\begin{aligned} \theta_3(Start) &= ((e, p_0)X)^* \\ \theta_3(X) &= (p Y)^*, (o Z)^*, (\vec{r} Y)^*, (\vec{r} Z)^*, (\vec{c} X)^* \\ \theta_3(Y) &= ((e, p_1)X)^*, ((e, p_2)X)^* \\ \theta_3(Z) &= ((e, p_3)X)^*, ((e, p_4)X)^*, (e, p_5)X! \end{aligned}$$

Note that the only change concerns X.

We now turn to introducing the constraint $\mathcal{C}_{\mathcal{G}}$. This constraint is needed in order to enforce that if an expression $((\tilde{e}_1, p_1)X_1, \dots, (\tilde{e}_k, p_k)X_k)^*$ is present in the ref-schema \mathcal{G} , then each p_1 node needs to be associated by means of a reference \vec{c} to a *unique* p_2 node which is his *brother* and which is in turn associated with a unique p_3 node, and so on. Note that, as desired, this constraint ensures that in each model, at a given level, for each i and j in $1 \dots k$, the number of p_i nodes is equal to that of p_j nodes.

Formally, we have:

Let $Exp_{\mathcal{G}}^*$ be the set of maximal sub-expressions of the form $((\tilde{e}_1, p_1)X_1, \dots, (\tilde{e}_k, p_k)X_k)^*$ occurring in \mathcal{G} . Then:

$$\mathcal{C}_{\mathcal{G}} = \bigwedge_{exp \in Exp_{\mathcal{G}}^*} \varphi_{exp}$$

Assuming exp is $((\tilde{e}_1, p_1)X_1, \dots, (\tilde{e}_k, p_k)X_k)^*$

$\varphi_{exp} = G^* \downarrow x \langle \tilde{e}_1 \rangle \psi_{exp}^{1,x}$, and, for $i = 1..k-1$,

$$\psi_{exp}^{i,x} = \downarrow y (p_i \wedge @_y \langle \vec{c} \rangle \downarrow z (p_{i+1} \wedge @_x \langle e_{j+1} \rangle z \wedge @_y [\vec{c}] z \wedge \psi_{exp}^{i+1,x}))$$

$$\psi_{exp}^{k,x} = \top$$

□

Example 5 Below, we give the constraint C_G for our running example. Recall that the simplified marked schema specified by the typing function θ_2 has two sub expressions of the form $((\tilde{e}_1, p_1)X_1)^*, \dots, ((\tilde{e}_k, p_k)X_k)^*$, namely:

$$\begin{aligned}\theta_2(Y) &= ((e, p_1)X, (e, p_2)X)^* \\ \theta_2(Z) &= ((e, p_3)X, (e, p_4)X)^*, (e, p_5)X\end{aligned}$$

Therefore we have

$$C_G = \phi_{((e, p_1)X, (e, p_2)X)^*} \wedge \phi_{((e, p_3)X, (e, p_4)X)^*}$$

with $\phi_{((e, p_i)X, (e, p_{i+1})X)^*}$ defined as

$$G^* \downarrow x \langle e \rangle \downarrow y (p_i \wedge @_y \langle \vec{c} \rangle) \downarrow z (p_{i+1} \wedge @_y [\vec{c}] z \wedge @_x \langle e \rangle z \wedge \top)$$

for $i = 1$ and $i = 3$.

Let us now summarize the translation of our running example. The normalized ref-schema \mathcal{G}_{norm} associated with the general ref-schema \mathcal{G} of example 2 is specified by the typing function θ_3 . The constraint C_G is given above. Intuitively, item 1 of Theorem 2 above tells that if \mathfrak{M} is a document conforming to the ref-schema \mathcal{G} , then there exists a document \mathfrak{M}_{norm} conforming to \mathcal{G}_{norm} and satisfying C_G . For the document \mathfrak{M} depicted in Figure 1, the document \mathfrak{M}_{norm} is given in Figure 2. The reader should be aware that in the presentation of \mathfrak{M}_{norm} , we did place the proposition p_i marking the node just next to the node. Thus he/she should not be confused by the fact that r is the name of the root as in Figure 1 and not a proposition.

Let us consider the left most "subtree" of the document of Figure 2. Note that the e leaves of this subtree could be marked in another manner as long as the number of nodes marked by p_1 is equal to the number of nodes marked by the proposition p_2 .

5 Constraint satisfiability under non recursive schemas

In our framework, both schemas and constraints are HML wffs which advantageously entails that the problem "does a document \mathfrak{M} conforming to the schema \mathcal{G} exist such that it satisfies the constraint C ?" can be restated directly as "is $\tau_G \wedge C$ finitely satisfiable?". This section is devoted to the presentation of a tableau proof system for testing satisfiability of $\tau_G \wedge C$. Obviously, the ultimate goal is to provide a terminating proof system. However, HML is not decidable in general [13, 22]. Thus we choose to consider several restrictions and relax, in a first step, finiteness over models.

Here we consider *non recursive normalized ref-schema* that is schemas that are not defining a non terminal symbol X by an expression using directly or indirectly X itself. We also do not consider markers in order to simplify the presentation. This restriction entails that, wrt accessibility relations associated with child labels, the depth of any instance of τ_G is bounded. Unfortunately, this is not sufficient to entail that it is finite. The next example exhibits a schema \mathcal{G} and a constraint C , such that $\tau_G \wedge C$ does not have the finite model property. Although found independently, this example is strongly related to the query given in [18] to show that the non positive fragment with left-sibling axis of Xpath does not have the finite model property.

Example 6 The schema is given by $\theta(Start) = (e End)^*$ and $\theta(End) = (\vec{e} End)^*$ and the constraint is $\psi_1 \wedge \psi_2 \wedge \psi_3 \wedge \psi_4$ where:

- ψ_1 is $\langle e \rangle \downarrow y (\langle \vec{e} \rangle y)$
- ψ_2 is $[e][\vec{e}] \downarrow y (@_{root} \langle e \rangle \downarrow z (\neg y \wedge @_y \langle \vec{e} \rangle z))$,
- ψ_3 is $[e] \downarrow x [\vec{e}][\vec{e}] \downarrow y @_x \langle \vec{e} \rangle y$
- ψ_4 is $[e] \downarrow x [\vec{e}] \downarrow y (@_x y \vee @_y [\vec{e}] \neg x)$.

Intuitively, the wff ψ_1 forces the instance to contain at least one e edge followed by a (reflexive) \vec{e} reference. The wff ψ_2 forces to create

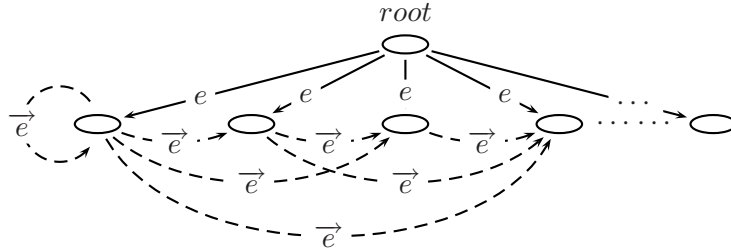


Figure 3: Infinite model

a new reference edge starting from the target of an existing reference. This is the main reason of the infiniteness of the model although it needs to be combined with the wffs ψ_3 and ψ_4 , expressing resp. the transitivity and antisymmetry over \vec{e} references. Indeed each model is isomorphic to the one given in Figure 3.

The satisfiability problem considered next is the following :

Input : a non recursive normalized ref-schema \mathcal{G}
a HML wff \mathcal{C} without F and G modalities

Problem : Is $\tau_{\mathcal{G}} \wedge \mathcal{C}$ satisfiable?

In the rest of the presentation, we abusively use the term instance of a schema \mathcal{G} to designate a potentially infinite model of $\tau_{\mathcal{G}}$.

The tableau system The tableau system defined below is geared to model building rather than refutation. This reversed use of tableau system is well-known [23]. Indeed the tableau system does not use directly the wff $\tau_{\mathcal{G}}$ but rather the wffs τ_{expr} as side effects of some of its rules. This has the advantage to make the rules for the modalities G and F unnecessary as long as they are not allowed in the constraint \mathcal{C} . Adding rule for these modalities is a subject of further study. Our tableau system is a prefixed tableau system: prefixes, here abusively called nominals⁵, are naming states

⁵We prefer not to use the term label for prefix as done in the literature.

and they are prefixing modal wffs. Roughly, the prefixed formula $n : \varphi$ intends to capture that during the proof, a state n has been created and that φ has to be satisfied at state n . This feature fits very well with modal logic [23, 3, 21] and particularly with HML [10]. Indeed, in a set of prefixed wffs Φ , the prefixed wffs of the form $n : \langle \vec{e} \rangle m$ can be seen as (the frame of) a Kripke model; in other words, it means that, the tableau deduction for Φ builds Kripke models of Φ internally while checking its satisfiability.

We assume wlg that the wff \mathcal{C} is closed rectified⁶ and in negation normal form⁷. The rules of the tableau system are given in the appendix. We provide an intuitive description of these rules below. The tableau rule deal with sets of (generalized) prefixed formulas; each rule is dedicated to one formation rule and thus a (generalized) prefixed formula is distinguished in the denominator of the rule. Generalized prefixed formula will be introduced when needed. The rules are partitioned in three groups: the propositional rules, the state variables and hybrid rules (these are the classical one [10, 12]) and finally the transition rules specified for the purpose of satisfiability checking in presence of non recursive schemas.

We define four transition rules. The $\langle E \rangle$ -rule below considers a distinguished prefixed wff of

⁶At most one binding $\downarrow x$ for each distinct state variable x .

⁷Negation only applied to propositions, nominals and state variables.

the form $\langle e \rangle \varphi$ where e a child label. It is a choice rule:

- the left part of the denominator propagates φ to an existing nominal (state) reachable by an existing e -edge,
- the right part expands the "model" with a new nominal (state) m and a new e -edge from n to m and propagates φ to m . The schema constraint $\tau_{\mathcal{G}}$ is locally enforced by introducing the prefixed wff $m : \tau_{type(e)}$.

The $\langle \vec{E} \rangle$ -rule below considers a distinguished prefixed wff of the form $\langle \vec{e} \rangle \varphi$ with \vec{e} a reference. It is a choice rule:

- the left part of the denominator is similar to that of the $\langle E \rangle$ -rule;
- the center part creates a new \vec{e} -reference from n to an existing state m while the condition $f \in child(\vec{e})$ ensures that m is well-typed, in other word it is meant to check that m is a valid destination for a \vec{e} -reference;
- the right part creates a new \vec{e} -reference edge leading to a new nominal (state) m . The wff $\pi(\vec{e}, m)$ enforces that m is linked to the root of the "model" by a well-typed path of child edges. This is one of the important features of our system which entails that a dandling state is never introduced. It is worth noticing that writing this formula is only possible because the schema \mathcal{G} is non recursive.

$$\pi(\vec{e}, m) =_{def} \bigvee_{e \in child(\vec{e})} \left(\bigvee_{ph \in Path(e)} \diamond(ph) m \right)$$

where $Path(e)$ is the set of paths in the dependency graph associated with \mathcal{G} whose source is $Start$ and last edge is labeled by e and where, if ph is the

path e_1, \dots, e_n , $\diamond(ph)$ is defined by $\langle e_1 \rangle \dots \langle e_n \rangle$.

The next two rules deal with the $[\tilde{e}]$ modality.

- The $[\mathcal{E}]$ -rule considers a distinguished prefixed formula of the form $[\tilde{e}] \varphi$ with \tilde{e} being any label. It generates an initial generalized prefixed formula further used to control that φ is propagated to all \tilde{e} -successors of n . A generalized prefix formula is of the form $n : ([\tilde{e}] \varphi, \Delta)$ where Δ stores the \tilde{e} successors of n over which the formula φ has already been propagated.
- The (Y) -rule processes new \tilde{e} -successors of n by propagating φ in order to enforce $[\tilde{e}] \varphi$ at state n .

Systematic construction of a \mathcal{G} -tableau **T** for \mathcal{C}

A \mathcal{G} -tableau **T** for \mathcal{C} is a proof tree, each branch corresponding to an attempt to build an instance of \mathcal{G} satisfying \mathcal{C} , hence a model of $\tau_{\mathcal{G}} \wedge \mathcal{C}$. The systematic construction follows more or less a breath-first traversal strategy with some priority over rule applications.

Stage 1 : $root : \tau_{Start} \wedge \mathcal{C}$ is the initial prefixed formula of the (root) of the tableau proof tree.

Stage $i+1$: Choose a leaf node **L** of the tableau under construction as close as possible to the root of the tableau. Assume that **L** contains a set Φ of formulas. Choose a (generalized) prefixed formula $n : \varphi$ in Φ in order to apply one of the tableau rules with the following priority:

- (1) propositional rules, state variable rule, hybrid rules,
- (2) $\langle E \rangle$ rules,
- (3) $\langle \vec{E} \rangle$ rules,
- (4) $[\mathcal{E}]$ and (Y) .

Expand **L** by applying the corresponding rule with respect to $n : \varphi$ in all manners meaning that if the rule is a choice rule, each possible application of the rule is developed, each one

leading to create a new descendant for L containing the appropriate prefixed formulas.

Proposition 1 (Fairness) If a (generalized) prefixed formula $n : \varphi$ belongs to some leaf L of the tableau under construction at stage i , then it will become the distinguished formula at some later step of the systematic construction.

Note that the systematic tableau construction may go on ad infinitum : the tableau constructed for the schema \mathcal{G} and constraint \mathcal{C} of Example 6 is infinite.

Definition 4 (Open/Closed \mathcal{G} -Tableau)

A \mathcal{G} -tableau T for the constraint \mathcal{C} is a (potentially infinite) tree constructed as described above. A branch⁸ \mathcal{B} is closed iff one of its nodes contains both prefixed wffs of the form $n : \varphi$ and $n : \neg\varphi$, or some statement $n : m$ for $n \neq m$. A branch is open if it is not closed and the tableau T is open iff one of its branches is open (otherwise it is closed).

Soundness and completeness of the tableau system

The proofs of soundness and completeness of the tableau system can be found in [9]. The proofs rest on the notion of \mathcal{G} -Hintikka set whose definition follows the line of and extends the one in [10]. Intuitively, a \mathcal{G} -Hintikka set is a potentially infinite set of prefixed formulas characterizing some model of $\tau_{\mathcal{G}}$ and thus in the proofs, \mathcal{G} -Hintikka sets bridge the gap between open branches of a \mathcal{G} -tableau T and models (potentially infinite instances of \mathcal{G}) of $\tau_{\mathcal{G}}$.

We now briefly discuss the main limitation of the tableau system presented in [9]: it is sound and complete for checking satisfiability, but not

⁸A branch of T is any path from the root downwards in the proof tree.

finite satisfiability which is of course the relevant notion for XML databases. As usual, two directions of investigation are possible: (1) the syntactic approach leads to exhibit conditions over the constraint \mathcal{C} in order to ensure the finite satisfiability, (2) the second direction is more “proof” oriented and involves enreaching the tableau system itself in order to prune infinite open branches. It turns out that investigating the first direction seems to lead to restrictions similar to that given in [28] where a new decidable fragment of HML is exhibited. Concerning the second direction, we are currently working at showing that the system can be enreached with a pruning test based on bisimulation. The claim is that, given S_1 and S_2 two proof segments such that S_2 extends S_1 , if their associated sets of prefixed formulas \mathcal{F}_1 and \mathcal{F}_2 are bisimilar then developing P_2 leads to an infinite branch. It is worth noticing that, proving the above claim would lead to show the decidability of finite satisfiability which is an open problem (see [5]).

6 Related works

Most of existing type languages for XML disregard the problem of typing references. Concerning type languages assuming sequence ordering, languages like DTD and all languages *a la* XDuce/CDuce [24, 6] do not permit to describe data with typed references. Concerning type languages not assuming sequence ordering, probably the most relevant one is the TQL language [16], which does not contain mechanisms to define references.

As already stated, besides our previous work [8], quite a few studies address typing mechanisms for references in semistructred and XML data. XML Schema [27] contains some mechanisms to type references. These mechanisms do not allow one to specify references and their target type in a flexible and direct manner.

Indeed, XML Schema uses XPath to specify typed references. As observed in [19], XPath is rather complex, requires a good amount of expertise to be used correctly and moreover, reasoning about constraints defined with XPath is highly intricate, if not impossible. XML Schema does not allow one to specify references whose target elements are possibly of different type. No mechanism is provided to define and check generic integrity constraints.

Simeon and Fan [19] propose an extension of DTD able to model classical relational and object oriented referential constraints. So the focus is on problems related to key constraints and foreign-key constraints. These constraints can be used to capture reference enforcement. It seems that the approach and that of XML are closely related. Rather negative results concerning decidability for key and foreign-key constraints have been showed in [4].

The quite recent paper [14] provides another interesting approach to the problem of logically characterizing XML schemas and constraints. Main differences wrt to that work, and references therein, are the followings. The work [14] proposes a decidable logic able to model inclusion constraints only over one attribute, otherwise decidability is lost. So, some kind of references can be modeled in the standard way by means of inclusions constraints. Of course decidability is lost if the references require multiple attributes. Here, instead of capturing references indirectly and partially like in [14], by means of inclusion constraints, we model typed references by using primitive mechanisms, defined by means of HML primitives and which are able to describe references in an abstract and general way, as much as for object-oriented databases. Also, differently from [14], the logic we consider allows to model navigational properties requiring to visit all descendants of certain nodes (by using $G\phi$ formulae); actually, the logic [14] essentially enables only one step

navigation (parent, child, ...) plus a kind of *somewhere* navigation, allowing to jump to some arbitrary node, not necessarily related with the current one. As stated in [14], if the proposed decidable logic is extended with navigational mechanisms like G , decidability is not proved to hold (it is an open problem). However, in our context, we are quite confident that this property holds for a wide class of constraints (involving F and G) under the assumption that data are constrained by a schema, and currently we are actively investigating this possibility.

Finally, a further difference wrt [14], and works referenced therein, is that while [14] deals with ordered XML documents, we move in a more database framework, where ordering is uninfluential.

7 Conclusions

To summarize, the aim of our study is to pave the way to a unique logical framework, allowing to deal with type and integrity constraints (as well as with queries) at the same time and in a clear, simple and flexible way. We would like to outline that the present work should not to be intended as yet another formalism to describe XML data with well-typed references, although ref-schemas are elegant extensions of DTDs. Indeed, this work presents basic techniques that can be easily extended in order to obtain HML logical characterisation of schemata defined according to existing techniques, like the two above cited ones, and of generic XML structural constraints. As stated in the introduction, this allows the use of existing techniques [9] to formally check important properties like constraint implication and constraint consistency in a *unique* framework, without the need of developing potential complex and error prone tools to cope with formalisms of different nature, like, say, DTD for

types and XPath for constraints [27].

One of the limitation of our framework is that, currently, we are not able to manage sequence ordering. This is not a strong limitation because the class of database applications where XML sequence ordering is non-influential is quite wide. However, HML seems able to describe sequence ordering as well, by using techniques similar to those we used to encode * types in the previous section.

Propositional rules: $(\alpha) \frac{n : \varphi \wedge \psi, \Phi}{n : \varphi, n : \psi, \Phi}$ $(\beta) \frac{n : \varphi \vee \psi, \Phi}{n : \varphi, \Phi \mid n : \psi, \Phi}$

State variable rule: $(Ref) \frac{\Phi}{n : n, \Phi}$ if n occurs in Φ

Hybrid rules : $(@) \frac{n : @_m \varphi, \Phi}{m : \varphi, \Phi}$ $(\downarrow) \frac{n : \downarrow x \varphi, \Phi}{n : \varphi[x \setminus n], \Phi}$

Propositional Rules, State variable rule, Hybrid rules

$$\frac{n : \langle e \rangle \varphi, \Phi}{\begin{array}{c|c} m : \varphi, \Phi & n : \langle e \rangle m, m : \tau_{type(e)}, m : \varphi, \Phi \\ \text{for } n : \langle e \rangle m \in \Phi & \text{for a new } m \end{array}}$$

Rules for the $\langle e \rangle$ modalities, where e is a child label

$$\frac{n : \langle \vec{e} \rangle \varphi, \Phi}{\begin{array}{c|c|c} m : \varphi, \Phi & n : \langle \vec{e} \rangle m, m : \varphi, \Phi & root : \pi(\vec{e}, m), root : \tau_{Start}, \\ & & m : \varphi, n : \langle \vec{e} \rangle m, \Phi \\ \text{for } & \text{for } p : \langle f \rangle m \in \Phi & \text{for a new } m \text{ and} \\ n : \langle \vec{e} \rangle m \in \Phi & \text{and } f \in child(\vec{e}) & \pi(\vec{e}, m) \text{ defined above} \end{array}}$$

Rules for the $\langle \vec{e} \rangle$ modalities, where \vec{e} is a reference

$$[\mathcal{E}] \frac{n : [\tilde{e}] \varphi, \Phi}{n : ([\tilde{e}] \varphi, \emptyset), \Phi} \quad (Y) \frac{n : ([\tilde{e}] \varphi, \Delta), \Phi}{\{m : \varphi \mid m \in \Delta'\}, n : ([\tilde{e}] \varphi, \Delta \cup \Delta'), \Phi}$$

for $\Delta' \neq \emptyset$ where
 $\Delta' = \{m \mid n : \langle \vec{e} \rangle m \in \Phi\} - \Delta$

Rules for the $[\tilde{e}]$ modalities, where \tilde{e} is any label

References

- [1] N. Alechina, S. Demri, and M. De Rijke. Path constraints from a modal logic point of view. In *Proc. of the 8th Int. Workshop on Knowledge Representation meets Databases*, 2001.
- [2] C. Areces, P. Blackburn, and M. Marx. Road-map on complexity for hybrid logics. In *Proc. of the 8th Annual Conf. of the EACSL*, pages 307–321. LNCS, Springer, 1999.
- [3] C. Areces, P. Blackburn, and M. Marx. Hybrid logics: Characterization, interpolation and complexity. *J. of Symbolic Logic*, 66(3):977–1010, 2001.
- [4] M. Arenas, W. Fan, and L. Libkin. Consistency of xml specifications. In *Inconsistency Tolerance*, pages 15–41, 2005.
- [5] M. Benedikt, W. Fan, and F. Geerts. Xpath satisfiability in the presence of dtDs. In *Proc. of the 24th Symp. on Principles of Database Systems*, pages 25–36. ACM SIGACT-SIGMOD-SIGART, 2005.
- [6] V. Benzaken, G. Castagna, and A. Frisch. Cduce: an xml-centric general-purpose language. In C. Runciman and O. Shivers, editors, *ICFP*, pages 51–63. ACM, 2003.
- [7] N. Bidoit, S. Cerrito, and V. Thion. A first step towards modeling semistructured data in hybrid multimodal logic. *J. of Applied Non-Classical Logics*, 14(4):447–476, 2004.
- [8] N. Bidoit, S. Cerrito, and V. Thion. A first step towards modeling semistructured data in hybrid multimodal logic. *Journal of Applied Non-Classical Logics*, 14(4):447–475, 2004.
- [9] N. Bidoit and D. Colazzo. Testing xml constraint satisfiability. Hylo, 2006.
- [10] P. Blackburn. Internalizing labelled deduction. *J. of Logic and Computation*, 10(1):137–168, 2000.
- [11] P. Blackburn. Representation, reasoning, and relational structures: a hybrid logic manifesto. *Logic J. of the IGPL*, 8(3):339–365, 2000.
- [12] P. Blackburn and M. Marx. Tableaux for quantified hybrid logic. In *Proc. Int. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 38–52, 2002.
- [13] P. Blackburn and J. Seligman. Hybrid languages. *J. of Logic, Language and Information*, 4:251–272, 1995.
- [14] 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.
- [15] P. Buneman, W. Fan, and S. Weinstein. Path constraints on semistructured and structured data. In *Proc. of the 17th Symp. on Principles of Database Systems*, pages 129–138. ACM SIGACT-SIGMOD-SIGART, 1998.
- [16] G. Conforti, G. Ghelli, A. Albano, D. Colazzo, P. Manghi, and C. Sartiani. The query language tql. In *WebDB*, pages 13–18, 2002.
- [17] S. Demri. (modal) logics for semistructured data. In *“Third Workshop on Methods for Modalities*. (Invited talk), 2003.
- [18] W. Fan and F. Geerts. Satisfiability of xpath queries with sibling axes. In *10th Int. Symp. on Database Programming Languages*, pages 122–137. LNCS, vol. 3774, 2005.

- [19] W. Fan and J. Siméon. Integrity constraints for xml. In *PODS*, pages 23–34. ACM, 2000.
- [20] M. Franceschet and M. De Rijke. Model checking for hybrid logics. In *Workshop Methods for Modalities*, 2003.
- [21] G. D. Giacomo and F. Massacci. Combining deduction and model checking into tableaux and algorithms for converse-PDL. *Information and Computation*, 162(1-2):117–137, 2000.
- [22] V. Goranko. Hierarchies of modal and temporal logics with reference pointers. *Journal of Logic, Language and Information*, 5(1):1–24, 1996.
- [23] R. Gore. Tableau methods for modal and temporal logics. Technical Report TR-ARP-15-95, Australian National University, Nov. 1995.
- [24] H. Hosoya and B. C. Pierce. Xduce: A typed xml processing language (preliminary report). In D. Suciu and G. Vossen, editors, *WebDB (Selected Papers)*, volume 1997 of *Lecture Notes in Computer Science*, pages 226–244. Springer, 2000.
- [25] S. Kripke. *Semantic Considerations on Modal Logic in Reference and Modality*. Oxford University Press, London, "1971".
- [26] M. Murata, D. Lee, and M. Mani. Taxonomy of xml schema languages using formal language theory. In *Proc. of the Extreme Markup Languages Conference*, 2001.
- [27] W. Recommendation. Xml schema. In <http://www.w3.org/TR/xmlschema-0/>.
- [28] B. ten Cate and M. Franceschet. On the complexity of hybrid logics with binders. In *CSL*, pages 339–354, 2005.