A Hybrid Logic for XML Reference Constraints

Carlo Combi, Andrea Masini, Barbara Oliboni, and Margherita Zorzi

Department of Computer Science – University of Verona, Ca' Vignal 2, Strada le Grazie 15, 37134 Verona, Italy

Abstract

XML emerged as the (meta) mark-up language for representing, exchanging, and storing semistructured data. The structure of an XML document may be specified either through DTD (Document Type Definition) language or through the specific language XML Schema. While the expressiveness of XML Schema allows one to specify both the structure and constraints for XML documents, DTD does not allow the specification of integrity constraints for XML documents. On the other side, DTD has a very compact notation opposed to the complex notation and syntax of XML Schema. Thus, it becomes important to consider the issue of how to express further constraints on DTD-based XML documents, still retaining the simplicity and succinctness of DTDs. According to this scenario, in this paper we focus on a (as much as possible) simple logic, named XHyb, expressive enough to allow the specification of the most common integrity and reference constraints in XML documents. In particular, we focus on constraints on ID and IDREF(S) attributes, which are the common way of logically connecting parts of XML documents, besides the usual parent-child relationship of XML elements. Differently from other previously proposed hybrid logics, in XHyb IDREF(S) attributes are explicitly expressible by means of suitable syntactical constructors. Moreover, we propose a refinement of the usual graph representation of XML documents in order to represent XML documents in a formal and intuitive way without flatten accessibility through IDREF(S) to the usual parent-child relationship. Model checking algorithms are then proposed, to verify that a given XML document satisfies the considered constraints.

Keywords: XML, DTD, constraints, hybrid logics

1. Introduction

The mark-up language XML (eXtensible Markup Language) is one of the main solutions used for creating and exchanging Web-based documents [1], for specifying different system configuration settings [2], for specifying metadata in many different, even web-based, applications [3, 4], and, in general, for representing and storing data and knowledge in a system-independent way [5].

XML allows the representation of structured and semistructured data through a hierarchical organization of mark-up elements [6]. An XML document is typically endowed with a DTD (Data Type Definition) [7]. DTDs allow the specification of the main structural features of XML documents in a simple and compact way. DTDs easily express hierarchies, order between elements, and several types of element attributes. In particular, the ID/IDREF(S) mechanism for DTD attributes describes identifiers and references. It has some similarities with both keys and foreign keys in a relational setting, and object identifiers and the related references between objects. Similarly to object identifiers (OIDs), ID attributes uniquely identify elements within the whole document. However, while OID values are hidden and managed directly by the system, ID attributes have a textual content, which is explicitly represented, and are managed by the user through explicit value assignments. This last feature is in the spirit of relational keys and foreign keys [8]. However, the values of ID attributes have to be different in an XML document and this is not true in relational DB where two keys can have different names, but same values. Thus, ID/IDREF(S) mechanism is in some sense in the middle between the semantics of relational keys/foreign keys and that of object-style references. Thus, the value of an attribute of type ID uniquely identifies an element node, while the value of an attribute of type IDREF(S) allows the reference to element node(s) on the base of values of its(their) ID attributes. For example, we may identify an element `patient` through an ID attribute `pat_id` with value "`pat01`", unique in the document. An element `therapy` would refer to the given patient, by an IDREF attribute `pat_ref` having the same value of `pat_id`.

DTD simplicity is paid in terms of expressiveness: a DTD efficiently models the structure of XML documents (it is able to provide a "syntactical" control such as context-free grammar), but it is not powerful enough for capturing more subtle, semantic features. As an example, (unique) values of ID attributes have the overall document as a scope. Consequently, attributes of type IDREF(S) cannot be constrained to refer to ID attribute values of a specific subset of element nodes (e.g., those of same element). On the other side, complex specification languages such as XML Schema [9] represent a powerful alternative

to DTD. XML Schema supports the specification of a very rich set of constraints (in terms of XPath expressions [10]) and seems to overcome DTD issues and limitations. Unfortunately, as observed in [11, 8, 12], XML Schema is too complicate and not compact at all in the specification of even simple integrity constraints.

Focusing on the research area related to XML modeling, in the literature there have been research efforts dealing with the proposal of formal approaches to specify different kinds of integrity constraints, trying to maintain the compactness of DTD specification [13, 14, 11, 15, 8, 16, 17, 18, 19]. Most efforts focused on formalisms allowing the expression of some kinds of *key and foreign key constraints*, by suitably extending DTD syntax and semantics [11, 16]. Complexity issues for the logical implication problem have been studied [8] as well as for the complexity of checking the consistency (or satisfiability) of different kinds of integrity constraints on XML documents [11, 19]. Other proposals deal with a comparison between DTD and XML Schema [20], introduce a new schema language simple as DTD but with the expressiveness of XML Schema [12], and study the implication problem and validation of keys in XML documents [21, 22, 23]. Moreover, some proposals deal with XML model checking problem, i.e., verifying whether a specific XML document is valid with respect to all the specified constraints [18]. Most proposals rely either on ad-hoc languages, e.g. the proposal in [8], or on logic-based languages, e.g. the proposal in [18], based on hybrid logics.

According to the above scenario, in this paper we focus on the issue of retaining in a logical framework the simplicity of DTDs with the capability of expressing meaningful integrity constraints. With respect to previous research efforts, we specifically propose a very simple formal language which is able to model constraints with respect to XML reference specification. More precisely, with some similarities with the approach in [18, 24], we propose a logical language, called XHyb, able to express *in a direct and explicit way* constraints on XML documents. The novelty of our work can be summarized as in the following.

- We propose XHyb, a new hybrid logics explicitly defined for specifying features of XML documents. XHyb allows us to encode constraints in terms of (as much as possible) simple modal formulas. Moreover, the Kripke-style XHyb models naturally fit the shape of XML documents, representing *explicitly and distinctly* both the parent-child relation and the reference specification for XML documents.
- We consider XML documents valid against a given DTD and do not extend DTD to allow further constraints (e.g., key constraints on ID attributes with a narrower scope than the entire document). We focus on the specification of ID/IDREF(S) reference constraints that cannot be expressed through DTD but that maintain the original validity of XML documents. We show how to encode interesting XML integrity constraints with an explicit focus on constraints related to ID/IDREF(S) mechanisms.
- We propose algorithms for XML model checking problem. More specifically, we present model checker XCheck, a simple, recursively defined algorithm, which implements the model checking problem in our theoretical setting.

A preliminary version of this work can be found in [25]. With respect to that version, here we characterize and describe XHyb in a deeper way, explicitly focused on XML document modeling. Moreover, the running example has been extended and enriched with new integrity constraints. Such constraints are discussed in an extensive way, by highlighting the differences with respect to previous contributions from literature and by also proposing some formula "templates" for the most common and recurring constraints. Finally, the XML model checking problem for XHyb is new to this paper.

The paper is structured as follows. In Section 2 we briefly recall main contributions presented in literature and show, focusing on reference constraints, the differences with respect to our system. In Section 3 we provide a motivating example that we will use throughout the paper to describe and clarify our proposal. Section 4 introduces both syntax and semantics of XHyb and discusses the correspondence between logical operators and XML constructs. In Section 5, we discuss how to translate in XHyb both XML documents and the constraints introduced in the motivating example. Model checker XCheck is defined in Section 6. Finally, Section 7 summarizes the main features of our proposal and sketches some future research lines.

2. Related Work

In this section we summarize the main issues faced in literature when dealing with XML constraints. In particular, we provide a description of the main kinds of XML constraints, and discuss the related solutions proposed in the literature, ending with a discussion on the complexity of the related checking algorithms. We also give an idea of our proposal by highlighting our contributions in the considered context.

Key constraints. In a DTD, it is possible to define ID attributes that allow one to uniquely identify a specific element node within an XML document. This means that the value of an ID attribute is unique with respect to the entire document, rather than among a specific given set of XML element nodes. Moreover, it is possible to specify at most one ID attribute for an element. Thus, using ID attributes as identifiers has some similarity with relational unary keys, as ID attributes are visible and could be related to the considered application domain. On the other hand, ID attributes have similarities with object identifiers in the object-oriented data representation, as they

have unique values within the overall document and their scope cannot be restricted by element/attribute name.

XML Schema allows the definition of key elements through XPath expressions. Starting from the XML Schema key proposal, in [13] and [14], the authors discuss the definition of keys for XML documents, with particular attention to the concepts of *absolute key* and *relative key*. A key can be considered either absolute or relative with respect to its scope: a key having the entire document as scope is absolute, while a key having a subpart of the document as scope is relative.

In [14], the authors show in theoretical terms, that automated reasoning about the fragment of XML keys with nonempty sets of simple key paths can be done efficiently. This fact has been confirmed in practice in [21], where the authors describe an efficient implementation of an algorithm deciding the implication problem for the considered expressive fragment of XML keys and thoroughly evaluate its performance. Moreover, in [17], the authors deal with the definition of XML keys by using path expressions. In particular, they consider XML keys that uniquely identify nodes of an XML tree by (complex) values on some selected descendant nodes. The contribution of the work is the definition of a new fragment of XML keys that keeps the right balance between expressiveness and efficiency of maintenance, by providing designers with an enhanced ability to capture properties of XML data that are significant for the application at hand.

In [12], the authors introduce an XML specification language, named BonXai, which is almost easy to use as DTD, but is able to incorporate many features of XML Schema, such as the use of types. As for key constraints, BonXai allows the user to express the same key constraints as XML Schema, in a simplified way. The main aim of this proposal is to overcome limitations of both DTD and XML Schema. On one side, BonXai tries to maintain the same level of DTD simplicity incrementing expressiveness, while, on the other side, it tries to maintain the same level of XML schema expressiveness, reducing the difficulties in its use.

In this paper we will focus on the specification of constraints for ID/IDREF(S) attribute references given a (standard) DTD declaration. According to this approach, we will not deal with issues related to the definition of (primary) key constraints by considering ad-hoc extensions for ID attributes (or for elements). Indeed, such constraints would imply the possibility of having XML documents not valid with respect to the corresponding standard DTD declaration. On the other hand, focusing on IDREF attributes, we will consider the capability of constraining one or multiple IDREF attributes to be key for a given element.

Foreign key constraints and more general integrity constraints. A foreign key constraint allows one to specify a relationship between two entities. In the context of relational databases, a foreign key is an attribute (or more attributes) in a relation that uniquely refers to a tuple of another relation (i.e., master table), through one of its keys. Foreign keys can be used to define integrity constraints, since they define a way for binding entities. The ID/IDREF(S) mechanism in DTDs provides a simple form of reference constraints. The reference to an ID attribute is defined by using IDREF/IDREFS attributes, which are untyped and thus do not allow the specification of the ID attribute to point to. This means that a DTD does not guarantee the control over ID/IDREF(S) references.

In [15], the authors propose a simple model of integrity constraints for XML. The proposal defines a single notion of keys and foreign keys based on a limited form of XPath expressions. In [8], the authors extend DTDs with several classes of integrity constraints such as keys and foreign keys, and more general ones. In particular, they propose several constraint languages for XML that provide both a reference mechanism and a semantics specification. It is worth noting that *our proposal does not deal with DTD extensions* considering typing and scoping of keys, as in [8]. Indeed, we will deal with standard DTDs, and propose XHyb to verify further desired properties on valid XML documents. We will give the possibility of enriching the definition of valid documents, that is, we will consider documents that are valid against a DTD and verify if a given document is valid also with respect to the specified constraints.

In [19], the authors focus on the specification of the structure of XML documents, by giving an abstract approach for the definition of classes of XML documents. The paper does not deal with the documents content, but it aims to state complex conditions about the documents structure, as, for example, the fact that if some element nodes appear in a document, then some other element nodes cannot appear in the same XML document.

The already mentioned language BonXai allows the expression of foreign key constraints, through the keyword *keyref*, similarly to XML Schema [12]. *Key* and *keyref* allow a simple definition of referential integrity constraints both for attribute and element content, possibly depending on the path of the given element/attribute (i.e., by specifying the scope of such constraints).

As for foreign key constraints, in this paper we will try to overcome the main limitations related to the use of ID/IDREF(S) attributes, considering DTD-based XML documents. Thus, we do not consider foreign key constraints that would imply some XML code not compliant with DTD (for example, DTD does not allow ID attributes to have a scope narrower than the entire document). Indeed, XHyb allows one, for example, to constrain an IDREF(S) attribute to refer to suitable ID attributes belonging to specific elements. In this sense, we will provide a mechanism for specifying the "type" the pointed ID must have. Further integrity constraints involving both ID/IDREF(S) attributes and parent-child relationships between elements will be discussed.

Cardinality constraints. Cardinality constraints can be specified in DTD by using a set of indicators (none, ?, +, *) specifying how many times an element will appear within an XML document. The considered indicators allow one to define general cardinalities, such as only once, zero or one, one or more, and zero or more.

In [26] and [17], the authors introduce soft cardinality constraints, where "soft" indicates that the constraints need to be satisfied on average only. The idea is to impose a constraint allowing some violations in order to consider the semistructured nature of XML document. This means that, soft cardinality constraints express a preferred situation, but allow violations of the corresponding strict constraints.

In this paper, we will consider cardinality constraints mainly related to element nodes linked through corresponding ID/IDREF(S) attributes.

Functional dependencies. Functional dependencies [27] (FDs) are crucial in database theory, and they have been widely studied in the context of relational databases. A functional dependency is a statement that describes a *semantics* constraint on data. FD theory has been extended to XML, through different proposals, in the same year by Lee and colleagues and then by Arenas and Libkin [28, 29, 30]. In [30], the authors introduce the concept of functional dependencies for XML, and define the semantics by using a relational representation of XML. FDs for XML documents allow one to express constraints as "`patient` elements, within root `patients`, having the same value for attribute `pathology` must have the same value for sub-element `division`". Moreover, they define an XML normal form avoiding update anomalies and redundancies. Authors' main goal is to find a way of converting an arbitrary DTD into a well-designed one that avoids these problems.

In our proposal we do not suppose to modify/convert a given DTD. Indeed, our approach deals with the definition and the verification of further reference constraints that cannot be expressed by DTDs. The only functional dependencies implicitly considered in our approach are those related to the introduced key properties of ID and IDREF attributes.

Checking XML Constraints. According to the goals and to the kinds of proposed constraints, different problems have been studied, related to XML constraint verification.

In [31], the authors discuss different kinds of reasoning problems for XML keys and provide some results for the satisfiability problem (i.e., given a finite set of keys, there exists an XML tree satisfying such keys?), and for the implication problem (i.e., deciding whether a new key holds given a set of known keys). In [11], the authors show that verifying the satisfiability of a DTD together with unary keys and foreign keys is NP-complete. The algorithm proposed and evaluated in [21] shows that the implication problem can be decided in quadratic time. The authors demonstrate that reasoning about expressive notions of XML keys can be done efficiently in practice and scales well. Similarly, considering FDs, in [30] the implication problem for FDs over simple DTDs is shown solvable in quadratic time.

A further basic problem is related to checking whether an XML document satisfies some given constraints. In [32], the authors discuss the performances of checking both XML inclusion dependency and XML foreign key, which are defined taking into account also the tree-structure of XML documents. They show that these kinds of constraints can be checked in linear time with respect to the number of tuples, i.e., those of atomic values composed by exploring the XML subtrees to analyze, and the number of root-based paths. Moving to key and key reference constraints, different proposals introduce algorithms working in linear time, with respect to the dimension of the checked XML document. In [33], a key validator is proposed, while in [34], the authors introduce a method for building an XML constraint validator from a given set of schemata, key and foreign key constraints. In [35], the authors address the problem of developing an efficient algorithm for checking whether an XML document satisfies a set of XML functional dependencies. The running time of the algorithm is linear in the size of the XML document and of the number of XML functional dependencies.

Moving to the problem of checking generic constraints on XML (and, more generally, semistructured) documents, some proposals reduce it to the *model checking* problem of some corresponding formula. In [18], the authors deal with the complexity of the model checking problem for hybrid logics. In particular, they establish a relationship between the query processing problem for semistructured and XML data and the global model checking problem for hybrid logics. In this context, they describe languages for specifying path constraints for semistructured data, including inclusion, inverse, and functional path constraints, and underline the close connection between path constraint evaluation for semistructured data and model checking for hybrid logics. They represent semistructured data as a graph in which a node corresponds to an object and an edge corresponds to an object attribute (i.e., an element name or an IDREF(S) attribute name). Edges are labelled with element names (or IDREF(S) attribute names), while leaf nodes are labelled with atomic string values (internal nodes are not labelled). They consider monadic queries, i.e., queries returning a set of nodes of the considered data graph. Queries in the most simple languages they consider have linear time data and expression complexity, while queries in the most complete language (with respect to well-known query languages for semistructured data) have exponential time expression complexity, i.e., with respect to the length of the query formula, and polynomial time data complexity, i.e., with respect to the dimension of the considered semistructured document. The constraint evaluation problem for key and functional constraints can be solved in linear time in the length of the constraint (ex-

pression complexity) and in polynomial time in the size of the semistructured document (data complexity).

In [36], the authors, after showing the connection between query languages applied to semistructured data and modal logics, have characterized families of queries (possibly corresponding to constraint expressions) that admit efficient solutions to the data retrieval problem. In particular, they consider graph-based queries and identify some families of them that correspond to CTL formulae. The problem of solving a query is thus considered as an instance of the model-checking problem for CTL that can be solved in polynomial time.

With respect to the proposed approaches, we will consider the problem of (model) checking an XML document against a given XHyb formula and will show that the complexity of the proposed checking algorithm is mainly related both to dimensions of formula and document, and to the number of (nested) quantifiers.

3. The Adopted XML model with a Running Example

In this paper we will adopt a simple XML model, usually related to DTD specifications. An XML document is represented as a set of nested elements, delimited by tags. Any XML document must have a single element containing all the other ones. Elements may contain other elements, a string, or may be empty. Elements may have attributes either mandatory or optional. We will mainly focus on ID and IDREF/IDREFS attributes. Such attributes are used to define further relationships between elements, beside the containment (i.e., parent-child) one. A DTD specifies the names of element tags, the content of elements, and relationships between elements through attributes. Often, an XML document is represented as a kind of node-labeled (ordered) tree. In Section 4.2 we will describe the specific tree-based representation we propose to represent DTD-based XML documents.

In this paper we will use the DTD shown in Figure 1 as a running example to discuss our proposal. The considered DTD describes a subset of data related to the university domain by defining relationships among students, professors, and courses. It represents the fact that a university is composed of many students, professors, courses, and examinations; a student may have a supervisor, when she starts her thesis work; a professor may act as both thesis supervisor and thesis reviewer.

The link between a student and her supervisor is modeled by using attribute `prof_ref` of type `IDREF` within element `supervisor` (which is contained in element `student`). On the other side, the corresponding link between a professor and her thesis students is modeled by means of attribute `stud_refs` of type `IDREFS` within element `thesis_stud`. Both attributes `supervisor` and `stud_refs` refer to elements identified by a suitable attribute of type `ID`.

```
<!ELEMENT university (student*,professor*,
course*,examination*)>
<!ELEMENT student (name,surname,supervisor?)>
<!ELEMENT professor (name,surname,thesis_stud?,
thesis_reviewer?)>
<!ELEMENT course (title)>
<!ELEMENT examination (grade, distinction?)>

<!ELEMENT name           (#PCDATA)>
<!ELEMENT surname        (#PCDATA)>
<!ELEMENT title          (#PCDATA)>
<!ELEMENT grade          (#PCDATA)>
<!ELEMENT distinction    (#PCDATA)>
<!ELEMENT supervisor      EMPTY>
<!ELEMENT thesis_stud     EMPTY>
<!ELEMENT thesis_reviewer EMPTY>

<!ATTLIST student         stud_id       ID     #REQUIRED>
<!ATTLIST professor       prof_id       ID     #REQUIRED>
<!ATTLIST course          cour_id       ID     #REQUIRED
prof_ref      IDREF  #REQUIRED>
<!ATTLIST examination     stud_ref      IDREF  #REQUIRED
cour_ref      IDREF  #REQUIRED>
<!ATTLIST supervisor      prof_ref      IDREF  #REQUIRED>
<!ATTLIST thesis_stud     stud_refs     IDREFS #REQUIRED>
<!ATTLIST thesis_reviewer stud_refs     IDREFS #REQUIRED>
```

Figure 1: An example of DTD for XML documents in a university domain

In general, DTD grammar allows us only to validate parent-child relationships (i.e., restrictions on the element structure of the document [16]) and links between `IDREF`/`IDREFS` values and `ID` values within the whole document. Thus, many obvious constraints cannot be explicitly modeled and some XML documents could be valid according to the given DTD but provide meaningless information, such as, for example, that a thesis reviewer is a course. Indeed, the DTD grammar does not allow us, for example, to constrain the value of attribute `prof_ref` to correspond to the value of attribute `prof_id` of some element `professor`.

Figure 2 depicts an XML document valid against the DTD reported in Figure 1.

Let us consider in the following some examples of requirements we would like to represent and verify in XML documents related to the university domain.

- A professor may supervise only students and students may be supervised only by professors. It means that, for example, a professor cannot be the supervisor of another professor (or of a course).
- A professor may be thesis reviewer of only students. Again, we want to avoid that a professor results to be thesis reviewer of a course (!).

Such constraints can be viewed as *foreign key constraints* as they require that the scope of some IDFREF(S) attribute values is restricted to the ID attribute values of some specific XML elements. Such scope restriction is not allowed in DTD documents, where we can only specify

```
<university>
  <student stud_id="stud1">
    <name> Al </name> <surname> Jones </surname>
    <supervisor prof_ref="prof1"/>
  </student>
  <student stud_id="stud2">
    <name> Sue </name> <surname> Storme </surname>
    <supervisor prof_ref="prof1"/>
  </student>
  <student stud_id="stud3">
    <name> Bill </name> <surname> Taylor </surname>
    <supervisor prof_ref="prof2"/>
  </student>
  <student stud_id="stud4">
    <name> Sam </name> <surname> Evans </surname>
  </student>
  <professor prof_id="prof1">
    <name> Ted </name> <surname> Wilson </surname>
    <thesis_stud stud_refs="stud1 stud2"/>
    <thesis_reviewer stud_refs="stud3"/>
  </professor>
  <professor prof_id="prof2">
    <name> May </name> <surname> West </surname>
    <thesis_reviewer stud_refs="stud1 stud2"/>
  </professor>
  <professor prof_id="prof3">
    <name> Ed </name> <surname> Smith </surname>
  </professor>
  <course cour_id="cour1" prof_ref="prof2">
    <title> Information Systems </title>
  </course>
  <course cour_id="cour2" prof_ref="prof1">
    <title> Computability </title>
  </course>
  <course cour_id="cour3" prof_ref="prof1">
    <title> Logics </title>
  </course>
  <course cour_id="cour4" prof_ref="prof3">
    <title> Databases </title>
  </course>
  <examination stud_ref="stud1" cour_ref="cour1">
    <grade>A</grade><distinction>+</distinction>
  </examination>
  <examination stud_ref="stud1" cour_ref="cour2">
    <grade>B</grade>
  </examination>
  <examination stud_ref="stud2" cour_ref="cour2">
    <grade>A</grade>
  </examination>
</university>
```

Figure 2: An XML document valid against the DTD in Figure 1

that IDREF(S) attribute values refer to ID attribute values within the overall XML document.

- A professor may be the thesis reviewer of up to three students.
- A student can be evaluated only once for a given course.

Such further *cardinality constraints* go beyond the DTD expressiveness as they require some counting mechanism in the first case and the capability of specifying a *key constraint* based on the of IDREF attributes `stud_ref` and `cour_ref` of element `examination`, in the second case.

- A professor cannot be both supervisor and reviewer of the same student.
- A professor can be supervisor only for students that attended and passed a course she taught.

These last *domain-related constraints* refer to the capability of expressing some requirements about to the specific application domain. They can be expressed only by leveraging the interplay between parent-child relationships and reference constraints.

4. XHyb: Hybrid Logic for XML Reference constraints

XHyb is an extension of a fragment of *hybrid logic* [37, 38, 39, 40]. Appendix A briefly sketches the main features of hybrid logic.

The specific features of XHyb are (i) the explicit representation of the tree-based structure of XML documents (by parent-child relation between elements), and (ii) the extension of quantified hybrid logic by means of a new modal operator $*$, which explicitly captures the presence of ID/IDREF(S) relation between elements of XML documents.

Let us suppose to have an XML document `D.xml`, valid against DTD document `D.dtd`.

We perform the following steps: (i) we extract a suitable hybrid language $\mathfrak{L}_{\mathtt{D}}$ based on `D.dtd`, and (ii) using the information in `D.xml`, we build an $\mathfrak{L}_{\mathtt{D}}$–structure, $\mathfrak{S}_{\mathtt{D}}$.

These two steps are related to the syntax and to the semantics of XHyb, respectively.

4.1. Syntax

4.1.1. Hybrid Language

An Hybrid Language $\mathfrak{L}_{\mathtt{D}}$ is a set of symbols as listed below:

- a finite set of *element names* $E_{\mathtt{D}} = \{e_0, \ldots, e_k\}$, given by the set of tag names in `D.dtd`;
- a finite set of *colors* $C_{\mathtt{D}} = \{c_0, \ldots, c_m\}$, given by the set of IDREF and IDREFS attributes in `D.dtd`;
- a denumerable set of *nominals* $\mathsf{N} = \{i_0, i_1, \ldots\}$. Nominals, as the main peculiarity of hybrid languages, univocally identify specific element nodes of the XML document;
- logical connectives and quantifiers $\rightarrow, \bot, \forall, *, @, \bigcirc$ and $\bullet$.

We assume that $C_{\mathtt{D}}$ and $E_{\mathtt{D}}$ are disjoint.

Example 1 *Let us consider DTD* `univ.dtd` *of Figure 1.* $\mathfrak{L}_{\mathtt{univ}}$ *has: (i) set* $E_{\mathtt{D}} = \{$`university`, `student`, `professor`, `course`, `examination`, `name`, `surname`, `supervisor`, `thesis_stud`, `thesis_reviewer`, `title`, `grade`, `distinction`$\}$*; (ii) set* $C_{\mathtt{D}} = \{$`prof_ref`, `stud_ref`, `cour_ref`, `stud_refs`$\}$.

In the following we will use symbols $p, q, \ldots$ for element names, $c, d, \ldots$ for color names, $i, j, k, l, m, n, \ldots$ for nominals.

We provide now the definition of well formed XHyb formulas.

4.1.2. Formulas

Given a language $\mathfrak{L}_{\mathtt{D}}$, the set of $\mathfrak{L}_{\mathtt{D}}$*–formulas* is the smallest set Y such that:

- $\mathsf{N} \subseteq Y$;
- $E_{\mathtt{D}} \subseteq Y$;
- if $i \in \mathsf{N}$ and $A \in Y$ then $(@_i A) \in Y$;
- if $i \in \mathsf{N}$ and $c \in C_{\mathtt{D}}$ then $*_c(i) \in Y$;
- if $i \in \mathsf{N}$ and $A \in Y$ then $(\forall i A) \in Y$;
- if $A, B \in Y$ then $(A \rightarrow B) \in Y$;
- $\bot \in Y$;
- if $A \in Y$ then $\bullet A, \bigcirc A \in Y$.

In the following we will use $A, B, \cdots$, possibly indexed, to range over the set of $\mathfrak{L}_{\mathtt{D}}$–formulas.

The intuition about the connectives is as follows:

- $@_i A$ means that formula A holds at element i. Following hybrid logic tradition, equality between two elements i and j is represented as $@_i j$;
- $*$ is the *reference operator*: $*_c(i)$ means that there is a reference, labelled by c, to the element with nominal i.
- $\forall$ is the usual universal quantifier of hybrid logics: $(\forall i A) \in Y$ means that formula A holds at each element i.
- $\rightarrow$ is the implication: $A \rightarrow B$ means that whenever formula A is true, then also formula B is true.
- $\bot$ is the basic symbol for falsehood.
- $\bullet A$ means that A holds at the parent of the given element, considering the parent-child relation in the tree structure of an XML document;
- $\bigcirc A$ means that A holds in each children of the given element;

In this way, we are distinctly modelling parent-child relation (by means of modal temporal operators) and reference accessibility. This perspective is further pursued in the definition of semantical models of the logic. This choice makes XHyb different from other hybrid systems developed for semistructured data representation and reasoning (see, for example [18, 24]).

In the rest of the paper, we will use the following (quite standard) abbreviations: $\neg A$ stands for $A \rightarrow \bot$; $A \vee B$ stands for $((\neg A) \rightarrow B)$; $A \wedge B$ stands for $\neg(A \vee B)$; $\exists i A$ stands for $\neg(\forall i(\neg A))$; $\bigcirc_{\exists} A$ stands for $\neg \bigcirc \neg A$. We will always omit the most external parentheses in formulas. Moreover we will adopt useful precedence between operators in order to simplify the readings of formulas, in particular we assume that $\neg, \forall, @, \bigcirc, \bullet$ have the higher priority. The only binder for variables is $\forall$. Therefore, the definition of the set of free variables in formulas is standard.

Example 2 *Some simple examples of* $\mathfrak{L}_{\mathtt{univ}}$*–formulas are as follows:*

- $@_i$`student` *means that formula* `student` *holds at the element identified by* i*, i.e. nominal* i *corresponds to an element* `student`*;*
- $*_{\mathtt{stud\_ref}}(i) \wedge @_i$`student` *means that there is a reference, labelled by* ***stud_ref****, to an element* ***student*** *having nominal* i*.*
- $\forall i(@_i$`university`$)$ *means that formula* `university` *holds at each element* i *(i.e., the XML document should contain only the root element* ***university****)).*
- `grade` $\rightarrow \bullet$`examination` *means that whenever formula* `grade` *is true, then also formula* `examination` *is true (i.e., a* ***grade*** *element is always contained in an element* ***examination****).*

As we will see in Sections 5 and 6, we focus on specific kinds of formulas, where all variables (i.e., nominals) correspond *(are bound) to a suitable quantifier. Thus, we need to distinguish *free* and *bound* variables.

Definition 3 (Free and bound variables) *The set* FV *of free variables for* $\mathfrak{L}_{\mathtt{D}}$*-formulas is inductively defined as follows:*

- $FV[i] = \{i\}$*;*
- $FV[@_i A] = \{i\} \cup FV[A]$*;*
- $FV[*_c(i)] = \{i\}$*;*
- $FV[\forall i A] = FV[A] - \{i\}$*;*
- $FV[A \rightarrow B] = FV[A] \cup FV[B]$*;*
- $FV[\bot] = \emptyset$*;*
- $FV[p] = \emptyset$ *for* $p \in \mathsf{PROP}$*;*
- $FV[\bullet A] = FV[A]$*;*
- $FV[\bigcirc A] = FV[A]$.

An occurrence of i *in an* $\mathfrak{L}_{\mathtt{D}}$*-formula* A *is bound iff there is a sub-*$\mathfrak{L}_{\mathtt{D}}$*-formula of* A *of the kind* $C = \forall i B$*. In this case we say also that* B *is the scope of* i*. We say that an occurrence of* i *in an* $\mathfrak{L}_{\mathtt{D}}$*-formula* A *is free iff it is not bound.*

Formulas without free variables are called *closed formulas*: for a closed formula A, $FV[A] = \emptyset$.

4.2. Semantics

We define now XHyb structures as an extension of Kripke-models to give a formal representation of XML documents and related DTDs. The idea is to decorate an *uncolored* tree (built upon document elements and the parent-child relationship) with *colored edges* representing references.

Definition 4 ($\mathfrak{L}_{\mathtt{D}}$-structure) *Given a language $\mathfrak{L}_{\mathtt{D}}$, an $\mathfrak{L}_{\mathtt{D}}$-structure $\mathfrak{S}_{\mathtt{D}}$ is a tuple:*

$$\mathfrak{S}_{\mathtt{D}} = \langle \mathcal{W}_{\mathtt{D}}, \mathcal{N}_{\mathtt{D}}, \mathfrak{r}_{\mathtt{D}}, \Game, V_E \rangle$$

where:

- $\langle \mathcal{W}_{\mathtt{D}}, \mathcal{N}_{\mathtt{D}}, \mathfrak{r}_{\mathtt{D}} \rangle$ *is a finite rooted tree, such that:*
  - $\mathcal{W}_{\mathtt{D}}$ *is the set of nodes corresponding to XML elements of document* `D.xml`*;*
  - $\mathcal{N}_{\mathtt{D}}$ *is the parent-child relationship. $(u, v) \in \mathcal{N}_{\mathtt{D}}$ means that element u is parent of (i.e., contains) element v in* `D.xml`*;*
  - $\mathfrak{r}_{\mathtt{D}} \in W$ *corresponds to the root element of* `D.xml`*;*
- $\Game$: $C_{\mathtt{D}} \times \mathcal{W}_{\mathtt{D}} \to 2^{\mathcal{W}_{\mathtt{D}}}$ *is a reference relation, where $C_{\mathtt{D}}$ is the finite set of colors in $\mathfrak{L}_{\mathtt{D}}$. $\Game\ (\mathtt{c}, w) = \{w' : w'$ has the attribute of type ID with value "*`a`*" and w has an IDREF/IDREFS attribute* `c` *="...*`a`*..."$\}$. Informally, given an attribute c of type IDREF/IDREFS and an element w, relation $\Game_{\mathtt{D}}$ yields the set of elements "accessible" from w through references contained in attribute* `c`*.*
- $V_E : E_{\mathtt{D}} \to 2^{\mathcal{W}_{\mathtt{D}}}$ *is an evaluation function for element names such that:*
  - $V_E(\mathtt{e}) = \{w : w$ *has element name* `e`$\}$

In the definition of XHyb we exploit the fact that the tree-like structure of XML documents naturally fits the shape of (most) modal/temporal logic Kripke models. This has been observed and used in [18, 24]. In this paper we started from the same observation, maintaining a slightly different viewpoint. Indeed, references have been dealt with independently from parent-child relationship: morally, the tree-component induced by the parent-child relationship is decorated by "colored" edges representing reference relations.

We can also notice that different fragments of the logic induce different shapes of the models, as depicted in Table 1.

Example 5 *Let us consider DTD* `univ.dtd` *in Figure 1, and document* `univ.xml` *in Figure 2. By performing the above steps it is easy to obtain the $\mathfrak{L}_{\mathtt{univ}}$–structure $\mathfrak{S}_{\mathtt{univ}}$. A graphical representation of $\mathfrak{S}_{\mathtt{univ}}$, explicitly considering both parent-child and reference relationships, is depicted in Figure 3.*

Table 2 summarizes the XML interpretation of XHyb, by showing the mapping between XHyb syntactical and semantic objects and the corresponding meaning in the XML document.

In analogy with first order logic, we define the concept of variable evaluation in $\mathfrak{L}_{\mathtt{D}}$-structure $\mathfrak{S}_{\mathtt{D}}$.

We need an evaluation function $\mathfrak{g}_{\mathfrak{S}_{\mathtt{D}}} : \mathsf{N} \to \mathcal{W}_{\mathtt{D}}$ that maps each nominal to a node of the XML-based structure.

When the underlying structure $\mathfrak{S}_{\mathtt{D}}$ is clear from the context we will omit the subscript in the evaluation function $\mathfrak{g}_{\mathfrak{S}_{\mathtt{D}}}$ and will write simply $\mathfrak{g}$.

Finally, we provide the definitions of satisfiability relation and satisfiability model, we will use in the following sections.

Definition 6 (Satisfiability relation) *Given a fixed $\mathfrak{L}_{\mathtt{D}}$-structure $\mathfrak{S}_{\mathtt{D}}$, an evaluation function $\mathfrak{g}$ and an element node $w \in \mathcal{W}_{\mathtt{D}}$, we inductively define the satisfiability relation*

$$\mathfrak{S}_{\mathtt{D}}, \mathfrak{g}, w \models A$$

by means of the following clauses:

- $\mathfrak{S}_{\mathtt{D}}, \mathfrak{g}, w \not\models \bot$*;*
- $\mathfrak{S}_{\mathtt{D}}, \mathfrak{g}, w \models p \Leftrightarrow w \in V_E(p)$ *with* $p \in E_{\mathtt{D}}$*;*
- $\mathfrak{S}_{\mathtt{D}}, \mathfrak{g}, w \models i \Leftrightarrow w = \mathfrak{g}(i)$*;*
- $\mathfrak{S}_{\mathtt{D}}, \mathfrak{g}, w \models *_c(i) \Leftrightarrow \mathfrak{g}(i) \in \Game\ (c, w)$*;*
- $\mathfrak{S}_{\mathtt{D}}, \mathfrak{g}, w \models @_i\ A \Leftrightarrow \mathfrak{S}_{\mathtt{D}}, \mathfrak{g}, \mathfrak{g}(i) \models A$*;*
- $\mathfrak{S}_{\mathtt{D}}, \mathfrak{g}, w \models @_i j \Leftrightarrow \mathfrak{g}(i) = \mathfrak{g}(j)$*;*
- $\mathfrak{S}_{\mathtt{D}}, \mathfrak{g}, w \models \forall i A \Leftrightarrow \forall v \in \mathcal{W}_{\mathtt{D}},\ \mathfrak{S}_{\mathtt{D}}, \mathfrak{g}[i \mapsto v], w \models A$*;*
- $\mathfrak{S}_{\mathtt{D}}, \mathfrak{g}, w \models A \to B \Leftrightarrow \mathfrak{S}_{\mathtt{D}}, \mathfrak{g}, w \not\models A$ ***or*** $\mathfrak{S}_{\mathtt{D}}, \mathfrak{g}, w \models B$*;*
- $\mathfrak{S}_{\mathtt{D}}, \mathfrak{g}, w \models \bullet A \Leftrightarrow \forall v \in \mathcal{W}_{\mathtt{D}}((v, w) \in \mathcal{N}_{\mathtt{D}}; \Rightarrow \mathfrak{S}_{\mathtt{D}}, \mathfrak{g}, v \models A)$*;*
- $\mathfrak{S}_{\mathtt{D}}, \mathfrak{g}, w \models \bigcirc A \Leftrightarrow \forall v \in \mathcal{W}_{\mathtt{D}}((w, v) \in \mathcal{N}_{\mathtt{D}} \Rightarrow \mathfrak{S}_{\mathtt{D}}, \mathfrak{g}, v \models A)$*.*

If $\mathfrak{S}_{\mathtt{D}}, \mathfrak{g}, w \models A$ we say that $\langle \mathfrak{S}_{\mathtt{D}}, \mathfrak{g}, w \rangle$ satisfies A.

If for each $w, \mathfrak{g}$ we have $\mathfrak{S}_{\mathtt{D}}, \mathfrak{g}, w \models A$ we write $\mathfrak{S}_{\mathtt{D}} \models A$ and we say that A is true in $\mathfrak{S}_{\mathtt{D}}$ or analogously that $\mathfrak{S}_{\mathtt{D}}$ is a model for A.

Let us now briefly focus on the semantics of XHyb particular connectives. The meaning of a formula $@_i A$ is defined by stipulating that A holds in an element node w if and only if $w = \mathfrak{g}(i)$, i.e., the interpretation by $\mathfrak{g}$ of nominal i is exactly w. The meaning of a formula $@_i j$ is defined by stipulating that the interpretation of nominals i and j is the same ($\mathfrak{g}$ maps both i and j to the same element node of the structure).

The meaning of a formula $\forall i A$ is given in the standard way, by using the notation $\mathfrak{g}[i \mapsto v]$. It specifies a function

Connectives	$\bullet, \bigcirc$	$*_c$	$*_c + \bullet, \bigcirc$
Relations	Parent-child	References	Parent-child + References
Shape of the models	tree component of $\mathfrak{S}_{\mathtt{D}}$	"colored" structure, i.e., nodes connected (only) by references	tree component of $\mathfrak{S}_{\mathtt{D}}$ +"colored edges"

Table 1: XHyb Overall Picture

XHyb **constructs**	**XML interpretation**
$\mathcal{W}_{\mathtt{D}}$ (Element nodes)	Element nodes occurring in XML document D
$E_{\mathtt{D}}$ (Element names)	Tag names declared in the DTD
$C_{\mathtt{D}}$ (Colors)	IDREF(S) attribute declared in the DTD
$V_E : E_{\mathtt{D}} \to 2^{\mathcal{W}_{\mathtt{D}}}$	Each element name e is mapped to the set of element nodes identifying occurrences of e
$\mathcal{N}_{\mathtt{D}} : \mathcal{W}_{\mathtt{D}} \to 2^{\mathcal{W}_{\mathtt{D}}}$	Parent-child relation
$\overline{\gamma}$: $\mathcal{W}_{\mathtt{D}} \to 2^{\mathcal{W}_{\mathtt{D}}}$	Each element node w is mapped to the set of element nodes w points to according to an IDREF(S) attribute (i.e., color)

Table 2: From XHyb to XML

$\mathfrak{g}[i \mapsto v] : \mathsf{N} \to \mathcal{W}_{\mathtt{D}}$ s.t. $\mathfrak{g}[i \mapsto v](x) = \mathfrak{g}(x)$ if $x \neq i$ and $\mathfrak{g}[i \mapsto v](x) = v$ if $x = i$.

The meaning of a formula $*_c(i)$ is defined upon the reference relation $\overline{\gamma}$. $*_c(i)$ holds in an element node w if and only if the interpretation by $\mathfrak{g}$ of nominal variable i belongs to the set of element nodes w points to, *through color* c, according to $\overline{\gamma}$. As previously said, the occurrence of color c in a XHyb formula represents the presence of an IDREF in a DTD and its semantics is given in terms of the reference relation.

Example 7 *Consider the XML document in Figure 2. The presence of an ID/IDREF relation between elements* `supervisor` *and* `professor` *can be easily encoded as* $\forall i(*_{\mathtt{prof\_ref}}(i) \rightarrow @_i\mathtt{professor})$. *This formula clearly holds at all element nodes* ***supervisor****, i.e., we can state* $\mathfrak{S}_{\mathtt{univ}} \models \forall i((\mathtt{supervisor} \wedge *_{\mathtt{prof\_ref}}(\mathtt{i})) \rightarrow @_i\mathtt{professor})$.

5. XHyb in action

XHyb syntax allows a simple and intuitive encoding of interesting XML constraints non-expressible by the DTD. In particular, we focus on foreign key constraints, cardinality constraints (implying some form of key constraints) and domain-related constraints. Foreign key constraints allow us to specify the elements having the ID attribute, an IDREF(S) attribute must refer to. Cardinality constraints allow the specification of the number of relationships instances that must be present through a specific ID/IDREF(S) mechanism. Moreover, a cardinality constraint may express a sort of key constraint for one or multiple IDREF attributes. Domain-related constraints represent more general requirements deriving from the modeled real-world context.

In this section we provide an XHyb encoding of some examples of constraints that must hold for the XML document in Figure 2. It is worth noting that, according to our approach, we will specify further constraints on valid XML documents and will not consider DTD-related constraints, as, for example, the fact that an IDREF attribute can contain only a value that must correspond to a value of an ID attribute in the XML document, or that IDREF and IDREFS attributes cannot have empty values.

5.1. Foreign key constraints

Let us consider the following requirements:

1. The supervisor of a student must be a professor.
2. A course must be taught by a professor.
3. An examination must be related to a student and to a course.

Considering the DTD declaration shown in Figure 1, the three above requirements may be expressed as: 1. Attribute `prof_ref` of element `supervisor` must refer to an element `professor`; 2. Attribute `prof_ref` of element `course` must refer to an element `professor`; 3. Attribute `stud_ref` of element `examination` must refer to an element `student` and, similarly, attribute `cour_ref` of element `examination` must refer to an element `course`.

All these constraints are foreign key constraints involving an IDREF attribute. Moreover, such attributes are within elements that must contain them. In other words,

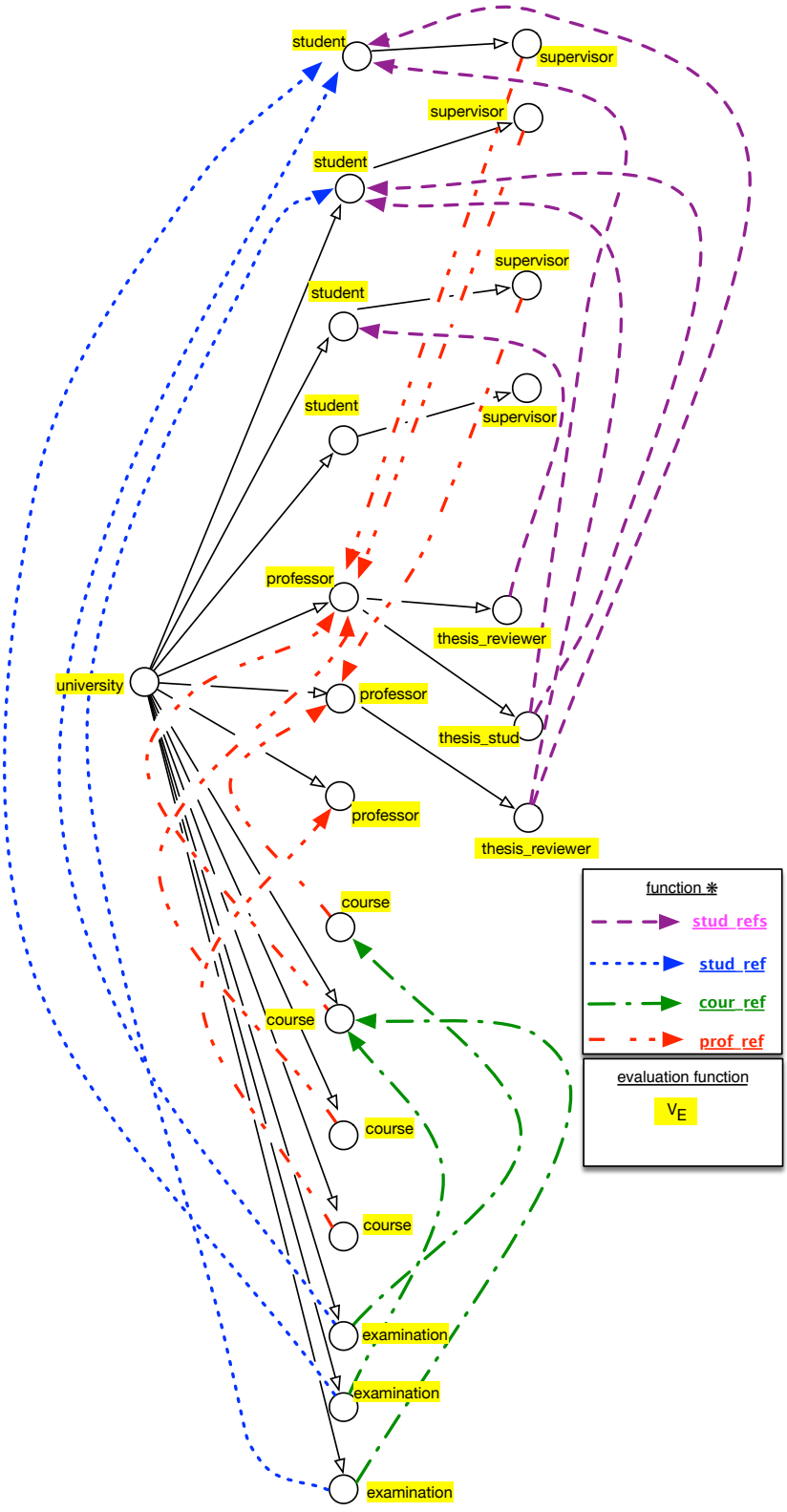

Figure 3: Graphical representation (references are represented through lines with different colors and shapes) of XML document in Figure 2. For sake of clarity, #PCDATA elements are not depicted.

and according to the considered DTD, such IDREF attributes have declared as `#REQUIRED`. Such kind of constraints can be expressed through some formulas having a fixed structure as:

$$\langle \texttt{ContainingE} \rangle \rightarrow \exists k(*_{\langle \texttt{IdrefA} \rangle}(k) \wedge @_k \langle \texttt{PointedE} \rangle)$$

where $\langle \texttt{ContainingE} \rangle$ stands for the element containing the considered IDREF attribute $\langle \texttt{IdrefA} \rangle$, and $\langle \texttt{PointedE} \rangle$ stands for the element containing the referenced ID attribute. Thus, we could say that the scope of attribute $\langle \texttt{IdrefA} \rangle$ of element $\langle \texttt{ContainingE} \rangle$ is restricted to values of ID attribute of element $\langle \texttt{PointedE} \rangle$. The proposed meta-formula applied to the three considered requirements corresponds to the following formulas, respectively:

1. $\texttt{supervisor} \rightarrow \exists k(*_{\texttt{prof\_ref}}(k) \wedge @_k \texttt{professor})$
2. $\texttt{course} \rightarrow \exists k(*_{\texttt{prof\_ref}}(k) \wedge @_k \texttt{professor})$
3. $\texttt{examination} \rightarrow (\exists k(*_{\texttt{cour\_ref}}(k) \wedge @_k \texttt{course}) \wedge \exists k(*_{\texttt{stud\_ref}}(k) \wedge @_k \texttt{student}))$

Let us now move to constraints that involve foreign key constraints expressed through IDREFS attributes and consider the following requirement:

4. A professor may be the supervisor of only students.
5. A professor may be thesis reviewer of only students.

According to the given DTD, when element `thesis_stud` appears, values of its attribute `stud_refs` must only refer to sets of ID attribute values of `student` element. Similarly to the previous case, when `thesis_reviewer` appears, its attribute `stud_refs` must only refer to sets of values of ID attribute of `student` element.

Such kind of constraints can be represented through the following (meta) formula:

$$\langle \texttt{ContainingE} \rangle \rightarrow \forall k(*_{\langle \texttt{IdrefsA} \rangle}(k) \rightarrow @_k \langle \texttt{PointedE} \rangle)$$

where $\langle \texttt{ContainingE} \rangle$ stands for the element containing the considered IDREFS attribute $\langle \texttt{IdrefsA} \rangle$, and $\langle \texttt{PointedE} \rangle$ stands for the element containing the referenced ID attribute. Thus, we could say that the scope of attribute $\langle \texttt{IdrefsA} \rangle$ of element $\langle \texttt{ContainingE} \rangle$ is restricted to set of values of ID attribute of element $\langle \texttt{PointedE} \rangle$.

Focusing on the introduced requirements, the following formulas specify the corresponding constraints in XHyb:

4. $\texttt{thesis\_stud} \rightarrow \forall k(*_{\texttt{stud\_refs}}(k) \rightarrow @_k \texttt{student})$
5. $\texttt{thesis\_reviewer} \rightarrow \forall k(*_{\texttt{stud\_refs}}(k) \rightarrow @_k \texttt{student})$

Such foreign key constraints can be further refined to contain also some requirements related to the structure of the XML document. As an example, assume that the DTD is changed as depicted in Figure 4, where only the modified/new parts are reported. The XML document is still valid according to the modified DTD. According to this DTD, we may have elements `student` contained within an element `not_paying_students`, corresponding to students who still have to pay the annual fee. Assume now that we want to express a more precise constraint.

```
<!ELEMENT university (student*,professor*,
course*,examination*,not_paying_students?)>
.....
.....
<!ELEMENT not_paying_students (student*)>
.....
 .....
```

Figure 4: An extension of the DTD for XML documents in a university domain

5.1 A professor may be thesis reviewer of only students, who already payed the annual fee.

Focusing on the introduced requirement, we need to explicitly specify that only elements `student` contained in `university` are allowed, while elements `student` contained in `not_paying_students` are not considered. The following formula specifies the above constraint in XHyb:

5.1 $\texttt{thesis\_reviewer} \rightarrow \forall k(*_{\texttt{stud\_refs}}(k) \rightarrow @_k(\texttt{student} \wedge \bullet\texttt{university}))$

5.2. Further integrity and cardinality constraints

We now discuss some more complex requirements involving integrity and cardinality constraints. Before going to requirements introduced in Section 3, let us focus on a common integrity constraint involving more than one ID/IDREF(S)-based references. Indeed, we could have some cross-references that need to be suitably managed in order to avoid meaningless data. Consider, for example, the following (implicit and obvious) requirement:

6. Students supervised by a professor compose the set of students doing the thesis work with the given professor.

According to the given DTD, when a `thesis_stud` element node appears, values of its attribute `stud_refs` must refer to sets of ID attribute values of `student` element nodes and such `student` element nodes must contain a `supervisor` element node having `prof_ref` attribute value referring to the ID attribute value of the `professor` element node containing the given `thesis_stud`.

Such kind of constraints can be expressed through some formulas having a fixed structure as:

$$\forall i(@_i(\langle \texttt{ContainingE} \rangle \rightarrow \forall k(*_{\langle \texttt{IdrefsA} \rangle}(k) \rightarrow @_k(\langle \texttt{PointedE} \rangle \wedge *_{\langle \texttt{IdrefB} \rangle}(\bullet i)))))$$

where $\langle \texttt{ContainingE} \rangle$, $\langle \texttt{IdrefsA} \rangle$, and $\langle \texttt{PointedE} \rangle$ have the same meaning as in the above formula, and $\langle \texttt{IdrefB} \rangle$

stands for the IDREF attribute "pointing back" to the ID attribute value of the parent node of the considered element node i. This kind of formulas can be slightly modified according to different choices in structuring and nesting elements. As an example, if the IDREFS attribute would be in the "main" element, and not in a nested empty one, $\bullet i$ would be simply replaced by i.

According to the introduced (meta) formula, the considered requirement can be expressed as in the following constraint:

6. $\forall i(@_i(\texttt{thesis\_stud} \rightarrow \forall k(*_{\texttt{stud\_refs}}(k) \rightarrow @_k(\texttt{student} \wedge *_{\texttt{supervisor}}(\bullet i)))))$

Let us now continue with other kinds of constraints and precisely with two different kinds of cardinality constraints.

7. A professor may be the thesis reviewer of up to 3 students.

8. A student can be evaluated only once for a given course.

According to the given DTD, the first requirement can be expressed as in the following. Any `thesis_stud` element node has attribute `stud_refs` containing up to three different references to ID attribute values of `student` element nodes.

Among the different ways of expressing such constraint, we will mainly consider elements `professor`, `student` and `supervisor`, together with attribute `prof_ref`, and will use the optional element `thesis_stud`, to avoid to consider the case of professors without any supervised student.

Let $\phi = @_j(\texttt{student} \wedge \bigcirc_{\exists}(\texttt{supervisor} \wedge *_{\texttt{prof\_ref}}(i)))$ be a (sub)formula with free variables j and i.

We can express the above constraint through the following formula

7. $(\forall i(@_i(\texttt{professor} \wedge \bigcirc_{\exists}\texttt{thesis\_stud}) \rightarrow ((\exists k_1, k_2, k_3(\phi[j/k_1] \wedge \phi[j/k_2] \wedge \phi[j/k_3]) \wedge \forall l(\phi[j/l]) \rightarrow (@_l k_1 \vee @_l k_2 \vee @_l k_3))))$

This formula uses subformula ϕ by binding variable i and by using variable j as a "parameter" substituted in the different parts of the formula with bounded variables k_1, k_2, k_3, and l. According to the formula, there can be at most three (possibly) different `student` element nodes pointing to the same `professor` element node. Moreover, any `student` element node pointing to the given `professor` element node must coincide with one of the three `student` element nodes.

As for the second requirement, according to the given DTD, attributes `stud_ref` and `cour_ref` are required to satisfy a *key constraint* on the couple of such IDREF attributes.

8. $\forall i \forall j((@_i\texttt{student} \wedge @_j\texttt{course}) \rightarrow \forall m \forall n(@_m(\texttt{examination} \wedge *_{\texttt{stud\_ref}}(i) \wedge *_{\texttt{cour\_ref}}(j)) \wedge @_n(\texttt{examination} \wedge *_{\texttt{stud\_ref}}(i) \wedge *_{\texttt{cour\_ref}}(j)) \rightarrow @_m n)$

It is simple to observe in this last formula both the *foreign key constraint* between `examination` attributes and `student` and `course` ID attributes, and the *key constraint* for the couple attributes `stud_ref` and `cour_ref`.

5.3. Domain-related constraints

Let us conclude with some specific requirements related to the considered application domain.

9. A professor cannot be both supervisor and reviewer of the same student.

10. A professor can be supervisor only for students that attended and passed a course she taught.

According to the given DTD, the first requirement can be expressed by saying that the `stud_refs` attribute value of a `thesis_stud` element node and the `stud_refs` attribute value of a `thesis_reviewer` element node must refer to two different and disjoint sets of ID values referring to `student` element nodes, when such nodes are child of the same `professor` element node. The following XHyb formula formally specifies the above constraint.

9. $\neg\exists k\ \exists j(@_k(\texttt{professor} \wedge \bigcirc_{\exists}(\texttt{thesis\_reviewer} \wedge *_{\texttt{stud\_refs}}(j)) \wedge \bigcirc_{\exists}(\texttt{thesis\_stud} \wedge *_{\texttt{stud\_refs}}(j)))))$

Finally, as for the second requirement, we could say that, given a `thesis_stud` element node, the values of its attribute `stud_refs` are forced to be a subset of the values of attributes `stud_ref` of `examination` element nodes, having the value of attribute `cour_ref` referring to a `course` element node having attribute `prof_ref` that refers to the `professor` element node containing the given `thesis_stud` element node.

10. $\forall i\ \forall k(@_i(\texttt{thesis\_stud} \wedge *_{\texttt{stud\_refs}}(k)) \rightarrow \exists m\ \exists n(@m(\texttt{course} \wedge *_{\texttt{prof\_ref}}(\bullet i)) \wedge @n(\texttt{examination} \wedge *_{\texttt{cour\_ref}}(m) \wedge *_{\texttt{stud\_ref}}(k))))$

The above formula formalises according to XHyb the above constraints. It is worth noting that some structural constraints remain implicit in the given formula, as, for example, the fact that element `thesis_stud` is child of `professor`, attribute `prof_ref` of element `course` points to.

6. XCheck: the XHyb Model Checker

XHyb is a suitable specification language for automatic constraint verification. As a first step toward automatic document validation, we define the Model Checking problem for XHyb.

The intuitive meaning of model checking is to establish whether an instance, i.e., a valid XML document together with its associated DTD, is a *model* for a formula. More formally, the model checking problem for a logic $\mathcal{L}$ on a class of structures $\mathcal{C}$ can be formulated as follows: given

a structure $\mathfrak{S} \in \mathcal{C}$, and a sentence $A \in \mathcal{L}$, decide if $\mathfrak{S}$ satisfies A.

The interest of model checking in the setting of semistructured data has been pointed out in [18], where authors study the computational complexity of the problem for a hybrid logic and provide some applications to XML constraint specification and query languages for semistructured data. Following the same motivation, in this section we use XHyb as a specification language and we define XCheck, a simple, recursively defined, model checking algorithm.

6.1. Adding element nodes as constants to the logic

In order to simplify the model checking procedure, we adopt here some (standard) modification to the syntax and semantics of XHyb (see, for example [41]). The idea is to eliminate (retaining an equivalent system) the need of evaluation function $\mathfrak{g}$, by introducing element nodes as constants in the syntax. This way, nominals in formulas stand for variables for element nodes and we can simplify the model checking algorithm we will provide.

Definition 8 (Extended logic) *The extended language $\mathfrak{L}_{\mathtt{D}}[\mathcal{W}_{\mathtt{D}}]$ is obtained by adding to $\mathfrak{L}_{\mathtt{D}}$ the set of constants representing specific element nodes of $\mathcal{W}_{\mathtt{D}}$.*

The definition of formulas and semantics is extended as follow (see [41] for the first-order case).

In the definition of formula add:

- $\mathcal{W}_{\mathtt{D}} \subseteq Y$;
- *if* $w \in \mathcal{W}_{\mathtt{D}}$ *and* $c \in C_{\mathtt{D}}$ *then* $*_c(w) \in Y$;
- *if* $w \in \mathcal{W}_{\mathtt{D}}$ *and* $A \in Y$ *then* $(@_w A) \in Y$.

In the definition of free variables add:

- $FV[w] = \emptyset$;
- $FV[*_c(w)] = \emptyset$.

The satisfiability relation is given only for closed formulas (i.e., without free variables). Indeed they represent different kinds of constraint, as exemplified in the previous section.

Definition 9 (Satisfiability relation) *Given a fixed $\mathfrak{L}_{\mathtt{D}}$-structure $\mathfrak{S}_{\mathtt{D}}$, and an element node $w \in \mathcal{W}_{\mathtt{D}}$ we inductively define satisfiability relation*

$$\mathfrak{S}_{\mathtt{D}}, w \models A$$

for A closed.

- $\mathfrak{S}_{\mathtt{D}}, w \not\models \bot$;
- $\mathfrak{S}_{\mathtt{D}}, w \models p \Leftrightarrow w \in V_E(p)$ *with* $p \in E_{\mathtt{D}}$;
- $\mathfrak{S}_{\mathtt{D}}, w \models w' \Leftrightarrow w = w'$;
- $\mathfrak{S}_{\mathtt{D}}, w \models *_c(w') \Leftrightarrow w' \in \overleftarrow{\forall}\,(c, w)$;
- $\mathfrak{S}_{\mathtt{D}}, w \models @_v A \Leftrightarrow \mathfrak{S}, v \models A$;
- $\mathfrak{S}_{\mathtt{D}}, w \models \forall i A \Leftrightarrow \forall v \in \mathcal{W}_{\mathtt{D}}, \mathfrak{S}_{\mathtt{D}}, w \models A[i/v]$[1];
- $\mathfrak{S}_{\mathtt{D}}, w \models \neg A \Leftrightarrow \mathfrak{S}_{\mathtt{D}}, w \not\models A$;
- $\mathfrak{S}_{\mathtt{D}}, w \models A \rightarrow B \Leftrightarrow \mathfrak{S}_{\mathtt{D}}, w \not\models A$ ***or*** $\mathfrak{S}_{\mathtt{D}}, w \models B$;
- $\mathfrak{S}_{\mathtt{D}}, w \models A \vee B \Leftrightarrow \mathfrak{S}_{\mathtt{D}}, w \models A$ ***or*** $\mathfrak{S}_{\mathtt{D}}, w \models B$;
- $\mathfrak{S}_{\mathtt{D}}, w \models A \wedge B \Leftrightarrow \mathfrak{S}_{\mathtt{D}}, w \models A$ ***and*** $\mathfrak{S}_{\mathtt{D}}, w \models B$;
- $\mathfrak{S}_{\mathtt{D}}, w \models \bullet A \Leftrightarrow \forall v \in \mathcal{W}_{\mathtt{D}}((v, w) \in \mathcal{N} \Rightarrow \mathfrak{S}_{\mathtt{D}}, v \models A)$;
- $\mathfrak{S}_{\mathtt{D}}, w \models \bigcirc A \Leftrightarrow \forall v \in \mathcal{W}_{\mathtt{D}}((w, v) \in \mathcal{N} \Rightarrow \mathfrak{S}_{\mathtt{D}}, v \models A)$.

Now we are ready to define (using a pseudocode) the model checker for XHyb, constraining ourselves to closed formulas in the language $\mathfrak{L}_{\mathtt{D}}[\mathcal{W}_{\mathtt{D}}]$.

6.2. The model checking problem

The model checking problem is formulated as follow. Given a closed formula A in the language $\mathfrak{L}_{\mathtt{D}}[\mathcal{W}_{\mathtt{D}}]$ and a $\mathfrak{L}_{\mathtt{D}}[\mathcal{W}_{\mathtt{D}}]$-structure $\mathfrak{S}_{\mathtt{D}}$, check algorithmically whether $\mathfrak{S}_{\mathtt{D}} \models A$.

Model checking is important in our approach since we propose XHyb as a concrete tool to validate XML document beyond DTD–validation. As already underlined, we assume here to have a DTD document `D.dtd` and an XML document `D.xml`, valid w.r.t. `D.dtd`. In this case, after the two steps we performed in building XHyb (see Section 4), i.e., (i) extracting a suitable hybrid language $\mathfrak{L}_{\mathtt{D}}$ based on `D.dtd`, and (ii) building an $\mathfrak{L}_{\mathtt{D}}$–structure $\mathfrak{S}_{\mathtt{D}}$ on the base of the content in `D.xml`, we perform three further steps: (iii) we extend the language with the set $\mathcal{W}_{\mathtt{D}}$ of element nodes of $\mathfrak{S}_{\mathtt{D}}$, obtaining the language $\mathfrak{L}_{\mathtt{D}}[\mathcal{W}_{\mathtt{D}}]$; (iv) we formulate a constraint as a closed formula A in $\mathfrak{L}_{\mathtt{D}}$; and (v) we check by means of suitable algorithms whether $\mathfrak{S}_{\mathtt{D}} \models A$. If the answer is "TRUE" we can state that document `D.xml` is valid w.r.t the constraint A, otherwise we say that `D.xml` is not valid w.r.t. constraint A.

Step (iii) implements the extension with constants for element nodes described in Section 6.1. Step (iv) requires the encoding of constraints according to the syntax of the extended logic.

In the following, we give the abstract (high level) procedures to perform step (v). We will use some auxiliary notations: with $\mathtt{pred}(w)$ and $\mathtt{succ}(w)$ we will denote the parent and the set of children of element node w, respectively.

[1] $A[i/v]$ is the new formula obtained by replacing each free occurrence of i in A with symbol v.

```
/* This is the top function. XCheck returns true if 𝔖_D ⊨ A,
   false otherwise. The procedure checks for each element
   node w whether 𝔖_D, w ⊨ A                                  */
fun XCheck(𝔖_D : structure, A : formula):bool
begin
    bool s ; foreach w ∈ 𝒲_D do
        s ← s ; and lXCheck(𝔖_D, A, w);
    return s ;
```

Algorithm 1: Main global procedure of the Model Checker

6.2.1. XCheck *abstract procedures*

The global procedure XCheck takes as inputs a $\mathfrak{S}$-structure and a formula A. It returns **true** if the structure satisfies A ($\mathfrak{S}_\mathtt{D} \models A$) and **false** otherwise. XCheck calls, on each element node in $\mathcal{W}_\mathtt{D}$, the *local* switch procedure *l*XCheck. *l*XCheck is recursively defined and calls the correct sub-routine according to the shape of the currently processed formula. Recursive callings on sub-formulas stop when propositional symbols are reached and then a boolean value is returned.

Main global and local procedures are reported in Algorithm 1 and 2. Subroutines for classical connectives and modal connectives are reported in Algorithm 3 and in Algorithm 4, respectively. Algorithm 5 contains subroutines for hybrid/reference connectives.

6.2.2. Correctness of the model checker

Let us now briefly discuss the correctness of algorithms we propose for model checking. The proof of correctness is mainly based on the correspondence of local procedure $l\mathsf{XCheck}(\mathfrak{S}_\mathtt{D}, A, w)$ with the definition of satisfiability relation according to the structure of formula A. Then, we move to prove the correctness of the global procedure, based on the local one.

Thus, we first sketch the correctness of $l\mathsf{XCheck}(\mathfrak{S}_\mathtt{D}, A, w)$.

Lemma 10 *lXCheck$(\mathfrak{S}_\mathtt{D}, A, w)$ returns true $\Leftrightarrow$ $\mathfrak{S}_\mathtt{D}, w \models A$*

Proof Sketch

We proceed by induction on the structure of A.

Base If $A \equiv p$ (i.e., we are in presence of an atomic formula and p is an element name)is atomic $l\mathsf{XCheck}(\mathfrak{S}_\mathtt{D}, p, w)$ returns *true* $\Leftrightarrow$ $w \in V_E(p)$ $\Leftrightarrow$ $\mathfrak{S}_\mathtt{D}, w \models p$

Inductive step

$A \equiv B \vee C$**:**
$l\mathsf{XCheck}(\mathfrak{S}_\mathtt{D}, B \vee C, w)$ returns *true*
$\Leftrightarrow$ (since the **switch** matches the **case** $B \vee C$)
$l\mathsf{XCheck}_\vee(\mathfrak{S}_\mathtt{D}, B, C, w)$ returns *true*
$\Leftrightarrow$
$l\mathsf{XCheck}(\mathfrak{S}_\mathtt{D}, B, w)$ returns *true*
or
$l\mathsf{XCheck}(\mathfrak{S}_\mathtt{D} C, w)$ returns *true*
$\Leftrightarrow$ (Induction Hypotesis)
$\mathfrak{S}_\mathtt{D}, w \models B$ or $\mathfrak{S}_\mathtt{D}, w \models C \Leftrightarrow \mathfrak{S}_\mathtt{D}, w \models B \vee C$

$A \equiv \neg B, A \equiv B \wedge C, A \equiv *_c(u), A \equiv @_v A$**:** proceed recursively in analogy with the previous case;

$A \equiv \forall i B$**:**
$l\mathsf{XCheck}(\mathfrak{S}_\mathtt{D}, \forall i B, w)$ returns *true*
$\Leftrightarrow$ (since the **switch** matches the **case** $\forall i B$)
$l\mathsf{XCheck}_\forall(\mathfrak{S}_\mathtt{D}, B, i, w)$ returns *true*
$\Leftrightarrow$ (execution of the command **foreach**)
for each $v \in \mathcal{W}_\mathtt{D}$
$l\mathsf{XCheck}(\mathfrak{S}_\mathtt{D}, B[i/v], w)$ returns *true*
$\Leftrightarrow$ (Induction Hypotesis)
$\forall v \in \mathcal{W}_\mathtt{D}, \mathfrak{S}_\mathtt{D}, w \models B[i/v] \Leftrightarrow$

```
/* The function considers the syntax of formula A. If A is a
   propositional symbol, the function verifies immediately it.
   Otherwise, it calls the suitable function depending on the
   logical operator to be checked.                            */
fun lXCheck(𝔖_D : structure, A : formula, w :
 element node):bool
begin
    switch A do
        case p do
            if w ∈ V_E(p) then
                return true;
            else
                return false;
        case u do
            if w = u then
                return true;
            else
                return false;
        case @_u B do
            return lXCheck_@(𝔖_D, B, u);
        case ◯B do
            return lXCheck_◯(𝔖_D, B, w);
        case ●B do
            return lXCheck_●(𝔖_D, B, w);
        case ∀iB do
            return lXCheck_∀(𝔖_D, B, i, w);
        case ∃iB do
            return lXCheck_∃(𝔖_D, B, i, w);
        case *_c(i) do
            return lXCheck_*(𝔖_D, c, i, w);
        case B ∨ C do
            return lXCheck_∨(𝔖_D, B, C, w);
        case B ∧ C do
            return lXCheck_∧(𝔖_D, B, C, w);
        case B → C do
            return lXCheck_→(𝔖_D, B, C, w);
        case ¬B do
            return lXCheck_¬(𝔖_D, B, w);
```

Algorithm 2: local procedure of the Model Checker

```
/* recursively call lXCheck on A                                   */
fun lXCheck¬(𝔖_D : structure, A : formula, u :
 element node):bool
begin
  return not lXCheck(𝔖, A, u);

/* Recursively call lXCheck on subformulas A and B                 */
fun lXCheck∨(𝔖_D : structure, A : formula, B :
 formula, u : element node):bool
begin
  return lXCheck(𝔖_D, A, u) or lXCheck(𝔖_D, B, u);
fun lXCheck∧(𝔖_D : structure, A : formula, B :
 formula, u : element node):bool
begin
  return lXCheck(𝔖_D, A, u) and lXCheck(𝔖_D, B, u);
fun lXCheck→(𝔖_D : structure, A : formula, B :
 formula, u : element node):bool
begin
  return not lXCheck(𝔖_D, A, u) or lXCheck(𝔖_D, B, u);

/* Recursively call procedure lXCheck on formulas A[i/w]
   for each possible renaming of free occurrences of nominal
   i with element nodes in 𝔖_D.                                   */
fun lXCheck∀(𝔖_D : structure, A : formula, i :
 nominal, u : element node):bool
begin
  bool s ← true;
  foreach w ∈ W_D do
    s ← s and lXCheck(𝔖_D, A[i/w], u);
  return s
fun lXCheck∃(𝔖_D : structure, A : formula, i :
 nominal, u : element node):bool
begin
  bool s ← true;
  foreach w ∈ W_D do
    s ← s and not lXCheck(𝔖_D, A[i/w], u);
    if s is false then
      return true;
  return false
```

Algorithm 3: Subroutines for classical connectives

```
/* Recursively call lXCheck on formula A for each successor
   element node s                                                  */
fun lXCheck◯(𝔖_D : structure, A : formula, u :
 element node):bool
begin
  bools ← true;
  foreach w ∈ succ(u) do
    s ← s and lXCheck(𝔖_D, A, w) ;
  return s;

/* Recursively call lXCheck on formula A and parent w              */
fun lXCheck●(𝔖_D : structure, B : formula, u :
 element node):bool
begin
  if u is the root then
    return true ;
  else
    return lXCheck(𝔖_D, B, pred(u));
```

Algorithm 4: Subroutines for modal connectives

```
/* test whether u belong to ▽(c, w)                                */
fun lXCheck*(𝔖_D : structure, u : element node, w :
 element node):bool
begin
  if u ∈▽(c, w) then
    return true;
  else
    return false;

/* Recursively call lXCheck with B and the new element
   node u                                                          */
fun lXCheck@(𝔖_D : structure, B : formula, u :
 element node):bool
begin
  return lXCheck(𝔖_D, B, u);
```

Algorithm 5: Subroutines for hybrid/reference connectives

$\mathfrak{S}_\mathsf{D}, \mathfrak{g}, w \models \forall i B$

$A \equiv \exists i B, A \equiv \bigcirc B, A \equiv \bullet B$**:** proceed recursively in analogy with the previous case

Now, we are ready to move to the correctness of the global procedure.

Theorem 11 $\mathsf{XCheck}(\mathfrak{S}_\mathsf{D}, A)$ *returns true* $\Leftrightarrow \mathfrak{S}_\mathsf{D} \models A$

Proof. Since $\mathsf{XCheck}(\mathfrak{S}_\mathsf{D}, A)$ returns *true* iff for each $w \in \mathcal{W}_\mathsf{D}$, $l\mathsf{XCheck}(\mathfrak{S}_\mathsf{D}, A, w)$ returns *true*, and (by definition) $\mathfrak{S}_\mathsf{D} \models A$ if and only if $\mathfrak{S}_\mathsf{D}, w \models A$, apply the previous lemma and conclude.

6.2.3. Complexity of XHyb *Model Checker*

In this section we sketch the complexity of XCheck, partially following the proof discussed in [18].

The complexity of the model checker depends on n and $\mathsf{size}(\mathsf{A})$, where $n = |\mathcal{W}_\mathsf{D}|$ and $\mathsf{size}(\mathsf{A})$ is the size of the checked formula, recursively defined as the number of symbols of A: $\mathsf{size}(\mathsf{p}) = \mathsf{size}(\mathsf{c}) = \mathsf{size}(\bot) = 1$; $\mathsf{size}(*_\mathsf{c}\mathsf{i}) = 2$; $\mathsf{size}(\forall \mathsf{i}\mathsf{A}) = \mathsf{size}(\mathsf{A})+2$; $\mathsf{size}(@_\mathsf{i}\mathsf{A}) = \mathsf{size}(\bigcirc\mathsf{A}) = \mathsf{size}(\bullet\mathsf{A}) = \mathsf{size}(\mathsf{A}) + 1$; $\mathsf{size}(\mathsf{A} \to \mathsf{B}) = \mathsf{size}(\mathsf{A}) + \mathsf{size}(\mathsf{B}) + 1$. Observe that $|\mathcal{N}| \in \mathcal{O}(n)$ and $|\,\triangledown(c, w)| \in \mathcal{O}(n)$ for all c and w in the given $\mathfrak{L}_\mathsf{D}$-structure.

To measure the complexity of the model checker, we proceed in two steps: (i) we analyze the model checking problem for the system XHyb^-, i.e. the fragment of XHyb without the universal quantification ($\forall$); (ii) we extend the result to the full XHyb.

Focusing on the auxiliary main function lXCheck, procedures for element names p, for element nodes u, and for formulas of the shape $*_c(w)$ require a constant amount of resources, in particular $\mathcal{O}(1)$ time units, respectively. Other procedures also involve a stack of recursive calls to subformulas. $\mathcal{O}(\mathsf{size}(\mathsf{A}))$ is clearly the upper bound for

the stack of recursive calls and also for the complexity of all procedures with the exception of $l\mathsf{XCheck}_{\bigcirc}$. $l\mathsf{XCheck}_{\bigcirc}$ is the most expensive subroutine for XHyb^{-} and requires $\mathcal{O}(\mathsf{size}(\mathsf{A}) \cdot n)$. It is easy to prove by induction that an arbitrary deep nesting of $\bigcirc$ operator ($\underbrace{\bigcirc \ldots \bigcirc}_{k} A$) always costs $\mathcal{O}(n)$ for any finite k and any model with finite size $n = |\mathcal{W}_{\mathtt{D}}|$.

Thus, the time complexity of the local model checker $l\mathsf{XCheck}$ for the fragment XHyb^{-} is $\mathcal{O}(\mathsf{size}(\mathsf{A}) \cdot n)$.

We are now ready to extend the proof to the full XHyb, adding the quantifier $\forall$.

For XHyb formula $\forall i.A$, the corresponding function $l\mathsf{XCheck}_{\forall}$ has complexity $n \cdot \mathcal{C}(A)$, where $\mathcal{C}(A)$ represent the complexity of subformula A. Moreover, along recursive calls, a nesting of universal quantifications may occur.

To provide a complexity bound, we introduce the concept of *nesting degree*, denoted as $\mathbf{d}_{\forall}(\cdot)$, related to nested universal quantifiers of a formula.

Definition 12 (Nesting degree) *The nesting degree $\mathbf{d}_{\forall}$ of an $\mathfrak{L}_{\mathtt{D}}$–formula is inductively defined as:*

- $\mathbf{d}_{\forall}(p) = \mathbf{d}_{\forall}(w) = \mathbf{d}_{\forall}(\bot) = \mathbf{d}_{\forall}(*_c i) = 0$;
- $\mathbf{d}_{\forall}(A \to B) = max(\mathbf{d}_{\forall}(A), \mathbf{d}_{\forall}(B))$;
- $\mathbf{d}_{\forall}(@_i A) = \mathbf{d}_{\forall}(\bigcirc A) = \mathbf{d}_{\forall}(\bullet A) = \mathbf{d}_{\forall}(A)$;
- $\mathbf{d}_{\forall}(\forall i A) = \mathbf{d}_{\forall}(A) + 1$.

The time complexity bound for $l\mathsf{XCheck}$ considering XHyb depends also on the nesting degree of the formula. At each step of the recursive call, in the worst case the switch function returns the subroutine $l\mathsf{XCheck}_{\forall}$, which, by definition, checks the current subformula at each element node of the XML document. The complexity of local procedure $l\mathsf{XCheck}$ for XHyb is thus $\mathcal{O}(\mathsf{size}(\mathsf{A}) \cdot n \cdot n^{\mathbf{d}_{\forall}(A)}) = \mathcal{O}(\mathsf{size}(\mathsf{A}) \cdot n^{\mathbf{d}_{\forall}(A)+1})$.

We can state now the overall result, taking into account that the top function XCheck calls the auxiliary main function $l\mathsf{XCheck}$ on each element node w in the structure.

Theorem 13 (Complexity of XCheck)
Let $\mathfrak{S}_{\mathtt{D}}$ be a $\mathfrak{L}_{\mathtt{D}}$-structure such that $n = |\mathcal{W}_{\mathtt{D}}|$ and let A be a XHyb formula.
$\mathsf{XCheck}(\mathfrak{S}_{\mathtt{D}}, A)$ *terminates in $\mathcal{O}(\mathsf{size}(\mathsf{A}) \cdot n^{\mathbf{d}_{\forall}(A)+2})$. Moreover, since the bound of recursive calls is $\mathcal{O}(\mathsf{size}(\mathsf{A}))$,* XCheck *requires polynomial space.*

6.3. Towards the implementation of XHyb*–structures*

Since the underlying data-structure of an XHyb–structure $\mathfrak{S}_{\mathtt{D}} = \langle \mathcal{W}_{\mathtt{D}}, \mathcal{N}_{\mathtt{D}}, \mathfrak{r}_{\mathtt{D}}, \Upsilon, V_E, V_R \rangle$ is the tree $\langle \mathcal{W}_{\mathtt{D}}, \mathcal{N}_{\mathtt{D}}, \mathfrak{r} \rangle$ we are in some sense forced to use a standard way to implement finite trees. This means to drop approaches based on adjacency matrices (tree are, de facto, scattered tree) in favour of approaches based on linked (non linear) lists.

The solution we propose here is quite simple, and is depicted in Figure 5, that shows the data structures needed to implement each element node.

Figure 6 shows the corresponding data type definitions in C syntax (it is easy to see how to implement the same concept in other languages that are able to manipulate lists, e.g. JAVA, ML-family an so on).

7. Conclusions and Future Work

In this paper we proposed a simple extension of hybrid logic with a reference operator $*_c$. We showed how this logic, called XHyb, is suitable to express reference constraints and thus overcomes, in an feasible way, some limitations of DTD expressiveness. Our work is open to several future directions. The first one is the proof-theoretic development of XHyb. It is possible to provide a full Hilbert-style axiomatization of the logic and to prove a Soundness and Completeness Theorem (the latter by means of a non-trivial but plain variation of Henkin's saturation technique for hybrid logic).

Moreover, we will study and develop extensions of the core logic presented in this paper, in order to model more sophisticated constraints, both improving expressiveness and retaining XHyb simplicity. The first step will be to capture arbitrary keys and foreign keys specification. Another interesting task will be the study of the XHyb expressive power w.r.t. a constraint taxonomy. Following the results proved in [42, 24], which relate Hybrid Logic, First Order Logic and XPath language, we aim to provide a complete account of the expressiveness of XHyb and its refinements.

Finally, we plan to implement the model checker algorithm XCheck defined in Section 6, as an important step toward the automatic constraint verification. Proposing and implementing an appropriate extension of the DTD language for the declaration of reference constraints, based on the provided XHyb formulas, is a further and interesting task.

References

[1] N. Bikakis, C. Tsinaraki, I. Stavrakantonakis, N. Gioldasis, S. Christodoulakis, The SPARQL2XQuery interoperability framework - utilizing schema mapping, schema transformation and query translation to integrate XML and the semantic web, World Wide Web 18 (2) (2015) 403–490. doi:10.1007/s11280-013-0257-x.
URL https://doi.org/10.1007/s11280-013-0257-x

[2] Y. Lu, J. Liang, Y. Xiao, S. Huang, D. Yang, W. Wang, H. Lin, XMLValue: XML configuration attribute value recommendation, in: S. Reisman, S. I. Ahamed, C. Demartini, T. M. Conte, L. Liu, W. R. Claycomb, M. Nakamura, E. Tovar, S. Cimato, C. Lung, H. Takakura, J. Yang, T. Akiyama, Z. Zhang, K. Hasan (Eds.), 41st IEEE Annual Computer Software and Applications Conference, COMPSAC 2017, Turin, Italy, July 4-8, 2017. Volume 1, IEEE Computer Society, 2017,

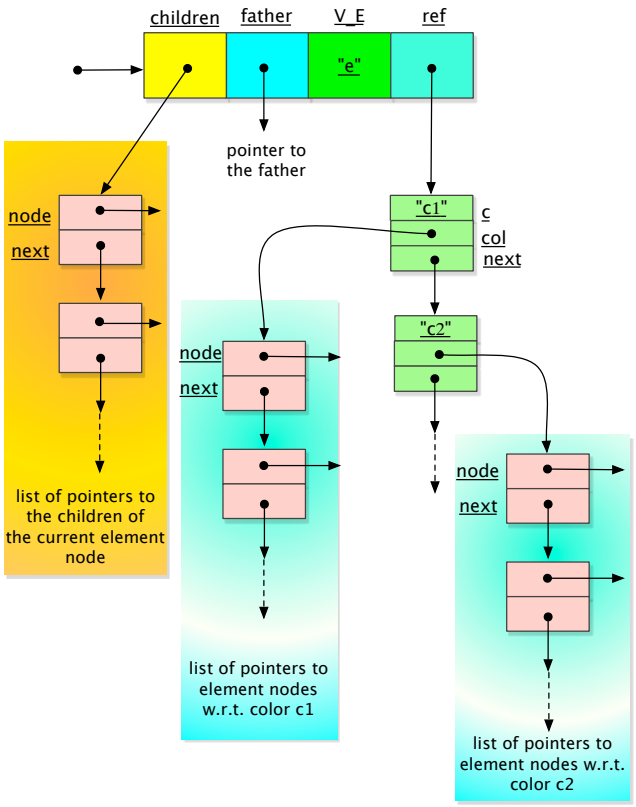

Figure 5: Graphical representation of the data structure for an element node

```
struct ElementNode{
    struct ElementNodeList * children; // the
        list of pointers to the children
    struct ElementNode * parent; // the pointer
        to the parent
    char V_E [100]; // the tag name of the
        element node (local implementation of
        function V|E)
    struct colors * ref; // the  reference
        relation (local implementation)
};
struct ElementNodeList{
    struct ElementNode * node; // children
    struct ElementNodeList * next;
};

struct colors{
    char [100] c; // a color
    struct ElementNodeList * col; //pointer to
        the list of element nodes for the color c
    struct colors * next;
};
```

Figure 6: The C–definition of the data-structures neeeded to represent XML element nodes

pp. 202–207. doi:10.1109/COMPSAC.2017.269.
URL https://doi.org/10.1109/COMPSAC.2017.269

[3] N. Palsetia, G. Deepa, F. A. Khan, P. S. Thilagam, A. R. Pais, Securing native XML database-driven web applications from xquery injection vulnerabilities, Journal of Systems and Software 122 (2016) 93–109. doi:10.1016/j.jss.2016.08.094.
URL https://doi.org/10.1016/j.jss.2016.08.094

[4] E. Chavarriaga, F. Jurado, F. Díez, An approach to build XML-based domain specific languages solutions for client-side web applications, Computer Languages, Systems & Structures 49 (2017) 133–151. doi:10.1016/j.cl.2017.04.002.
URL https://doi.org/10.1016/j.cl.2017.04.002

[5] J. Tekli, N. Charbel, R. Chbeir, Building semantic trees from XML documents, J. Web Sem. 37-38 (2016) 1–24. doi:10.1016/j.websem.2016.03.002.
URL https://doi.org/10.1016/j.websem.2016.03.002

[6] World Wide Web Consortium. Extensible Markup Language (XML), https://www.w3.org/XML/ (2016).

[7] World Wide Web Consortium. W3C XML specification DTD, https://www.w3.org/XML/1998/06/xmlspec-report-19980910.htm (1998).

[8] W. Fan, J. Siméon, Integrity constraints for XML, J. Comput. Syst. Sci. 66 (1) (2003) 254–291. doi:10.1016/S0022-0000(02)00032-6.

[9] E. van der Vlist, XML Schema - the W3C's object-oriented descriptions for XML., O'Reilly, 2002.

[10] XML path language (XPath) Version 1.0, https://www.w3.org/TR/xpath/, w3C Reccomendation 2016.

[11] W. Fan, L. Libkin, On XML integrity constraints in the presence of DTDs, Journal of ACM 49 (3) (2002) 368–406. doi:10.1145/567112.567117.

[12] W. Martens, F. Neven, M. Niewerth, T. Schwentick, Bonxai: Combining the simplicity of DTD with the expressiveness of XML schema, ACM Trans. Database Syst. 42 (3) (2017) 15:1–15:42. doi:10.1145/3105960.
URL http://doi.acm.org/10.1145/3105960

[13] P. Buneman, S. Davidson, W. Fan, C. Hara, W.-C. Tan, Keys for XML, Computer Networks 39 (5) (2002) 473 – 487. doi:10.1016/S1389-1286(02)00223-2.

[14] P. Buneman, S. Davidson, W. Fan, C. Hara, W.-C. Tan, Reasoning about keys for XML, Information Systems 28 (8) (2003) 1037 – 1063. doi:http://dx.doi.org/10.1016/S0306-4379(03)00028-0.

[15] W. Fan, G. M. Kuper, J. Siméon, A unified constraint model for XML, Computer Networks 39 (5) (2002) 489 – 505. doi:http://dx.doi.org/10.1016/S1389-1286(02)00219-0.

[16] W. Fan, XML constraints: Specification, analysis, and applications, in: 16th International Workshop on Database and Expert Systems Applications (DEXA 2005), 22-26 August 2005, Copenhagen, Denmark, IEEE Computer Society, 2005, pp. 805–809. doi:10.1109/DEXA.2005.204.
URL https://doi.org/10.1109/DEXA.2005.204

[17] F. Ferrarotti, S. Hartmann, S. Link, Efficiency frontiers of XML cardinality constraints, Data Knowl. Eng. 87 (2013) 297–319.

[18] M. Franceschet, M. de Rijke, Model checking hybrid logics (with an application to semistructured data), Journal of Applied Logic 4 (3) (2006) 279 – 304.

[19] M. Navarro, F. Orejas, E. Pino, Satisfiability of constraint specifications on XML documents, in: Logic, Rewriting, and Concurrency - Essays dedicated to José Meseguer on the Occasion of His 65th Birthday, Vol. 9200 of Lecture Notes in Computer Science, Springer, 2015, pp. 539–561. doi:10.1007/978-3-319-23165-5_25.

[20] G. J. Bex, F. Neven, J. V. den Bussche, DTDs versus XML schema: A practical study, in: S. Amer-Yahia, L. Gravano (Eds.), Proceedings of the Seventh International Workshop on the Web and Databases, WebDB 2004, June 17-18, 2004, Maison de la Chimie, Paris, France, Colocated with ACM SIGMOD/PODS 2004, 2004, pp. 79–84.
URL http://webdb2004.cs.columbia.edu/papers/6-1.pdf

[21] F. Ferrarotti, S. Hartmann, S. Link, M. Marín, E. Muñoz, Performance analysis of algorithms to reason about XML keys, in: S. W. Liddle, K. Schewe, A. M. Tjoa, X. Zhou (Eds.), Database and Expert Systems Applications - 23rd International Conference, DEXA 2012, Vienna, Austria, September 3-6, 2012. Proceedings, Part I, Vol. 7446 of Lecture Notes in Computer Science, Springer, 2012, pp. 101–115. doi:10.1007/978-3-642-32600-4_9.
URL https://doi.org/10.1007/978-3-642-32600-4_9

[22] F. Ferrarotti, S. Hartmann, S. Link, M. Marín, E. Muñoz, The finite implication problem for expressive XML keys: Foundations, applications, and performance evaluation, Trans. Large-Scale Data- and Knowledge-Centered Systems 10 (2013) 60–94. doi:10.1007/978-3-642-41221-9_3.
URL https://doi.org/10.1007/978-3-642-41221-9_3

[23] B. Liu, S. Link, E. Muñoz, Validation of expressive XML keys with XML Schema and XQuery, in: M. Saeki, H. Köhler (Eds.), 11th Asia-Pacific Conference on Conceptual Modelling, APCCM 2015, Sydney, Australia, January 2015, Vol. 165 of CRPIT, Australian Computer Society, 2015, pp. 81–90.
URL http://crpit.com/abstracts/CRPITV165Liu.html

[24] M. Marx, Xpath and modal logics of finite DAG's, in: Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2003, Rome, Italy, September 9-12, 2003. Proceedings, 2003, pp. 150–164.

[25] C. Combi, A. Masini, B. Oliboni, M. Zorzi, A logical framework for XML reference specification, in: Q. Chen, A. Hameurlain, F. Toumani, R. R. Wagner, H. Decker (Eds.), Database and Expert Systems Applications - 26th International Conference, DEXA 2015, Valencia, Spain, September 1-4, 2015, Proceedings, Part II, Vol. 9262 of Lecture Notes in Computer Science, Springer, 2015, pp. 258–267. doi:10.1007/978-3-319-22852-5_22.
URL https://doi.org/10.1007/978-3-319-22852-5_22

[26] F. Ferrarotti, S. Hartmann, S. Link, M. Marín, E. Muñoz, Soft cardinality constraints on XML data - how exceptions prove the business rule, in: X. Lin, Y. Manolopoulos, D. Srivastava, G. Huang (Eds.), Web Information Systems Engineering - WISE 2013 - 14th International Conference, Nanjing, China, October 13-15, 2013, Proceedings, Part I, Vol. 8180 of Lecture Notes in Computer Science, Springer, 2013, pp. 382–395. doi:10.1007/978-3-642-41230-1_32.

[27] E. F. Codd, A relational model of data for large shared data banks, Commun. ACM 13 (6) (1970) 377–387. doi:10.1145/362384.362685.

[28] M.-L. Lee, T. W. Ling, W. L. Low, Designing functional dependencies for XML, in: Proceedings of the 8th International Conference on Extending Database Technology: Advances in Database Technology, EDBT '02, Springer-Verlag, London, UK, UK, 2002, pp. 124–141.

[29] M. Arenas, L. Libkin, A normal form for XML documents, in: L. Popa, S. Abiteboul, P. G. Kolaitis (Eds.), Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA, ACM, 2002, pp. 85–96. doi:10.1145/543613.543625.
URL http://doi.acm.org/10.1145/543613.543625

[30] M. Arenas, L. Libkin, A normal form for XML documents, ACM Trans. Database Syst. 29 (1) (2004) 195–232.

[31] P. Buneman, S. B. Davidson, W. Fan, C. S. Hara, W. C. Tan, Reasoning about keys for XML, Inf. Syst. 28 (8) (2003) 1037–1063. doi:10.1016/S0306-4379(03)00028-0.
URL https://doi.org/10.1016/S0306-4379(03)00028-0

[32] M. S. Shahriar, J. Liu, Checking satisfactions of XML referential integrity constraints, in: J. Liu, J. Wu, Y. Yao, T. Nishida (Eds.), Active Media Technology, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 148–159.

[33] Y. Chen, S. B. Davidson, Y. Zheng, XKvalidator: a constraint validator for XML, in: Proceedings of the 2002 ACM CIKM International Conference on Information and Knowledge Management, McLean, VA, USA, November 4-9, 2002, ACM, 2002, pp. 446–452. doi:10.1145/584792.584866.
URL http://doi.acm.org/10.1145/584792.584866

[34] M. A. Abrão, B. Bouchou, M. H. Ferrari, D. Laurent, M. A. Musicante, Incremental constraint checking for XML documents, in: Z. Bellahsène, T. Milo, M. Rys, D. Suciu, R. Unland (Eds.), Database and XML Technologies, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 112–127.

[35] M. W. Vincent, J. Liu, Checking functional dependency satisfaction in XML, in: S. Bressan, S. Ceri, E. Hunt, Z. G. Ives, Z. Bellahsène, M. Rys, R. Unland (Eds.), Database and XML Technologies, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 4–17.

[36] A. Dovier, E. Quintarelli, Applying model-checking to solve queries on semistructured data, Computer Languages, Systems & Structures 35 (2) (2009) 143 – 172. doi:https://doi.org/10.1016/j.cl.2006.11.002.
URL http://www.sciencedirect.com/science/article/pii/S1477842406000339

[37] P. Blackburn, Representation, reasoning, and relational structures: a hybrid logic manifesto, Logic Journal of the IGPL 8 (3) (2000) 339–365.

[38] P. Blackburn, B. Ten Cate, Pure extensions, proof rules, and hybrid axiomatics, Studia Logica 84 (2) (2006) 277–322.

[39] P. Blackburn, M. Tzakova, Hybridizing concept languages, Annals of Mathematics and Artificial Intelligence 24 (1-4) (1998) 23–49.

[40] A. N. Prior, Past, Present and Future, Clarendon, Oxford, 1967.

[41] D. van Dalen, Logic and Structure, Universitext (1979), Springer-Verlag, London, UK, 2013. doi:10.1007/978-1-4471-4558-5.

[42] C. Areces, P. Blackburn, M. Marx, Hybrid logics: Characterization, interpolation and complexity, J. Symb. Log. 66 (3) (2001) 977–1010. doi:10.2307/2695090.

Appendix A. Hybrid Logics

Hybrid logics are a class of logics extending classical modal/temporal logics.

It is well known that a propositional modal logic with necessity operator $\Box$ (actually, in the paper we have two kinds of necessity operators, $\bigcirc$ and $\bullet$) is modeled by means of a so called Kripke model, namely a structure of the kind $M = \langle W, R, V\rangle$ where W is a set of so-called *possible worlds*, $R \subseteq W \times W$ is called *accessibility relation* (in the paper we have more complex structures; in any case for this short survey, graph structures $\langle W, R\rangle$ suffice), and $V : Prop \to 2^W$ is a function assigning to each propositional p symbol the set of worlds where p is true.

Once we have fixed a Kripke model M, the semantics of a formula is given by means of the so-called Kripke semantics. For example, a formula $\Box A$ is true in the world w iff A is true in all the accessible worlds from w:

$$M, w \models \Box A \Leftrightarrow \text{ for all } u : wRu, u \models A$$

The main idea behind Hybrid logics is the introduction in the logical language of explicit references to possible worlds. The basic step is given by enriching the language with a new syntactical category, i.e., the so-called *nominals*. A nominal i is a kind of variable referring to one possible world w. Semantically, the connection between nominals and possible worlds is realized by means of an evaluation function $g : N \to W$, which assigns to each i a world $g(i) \in W$.

We say that a nominal i is true with regard to a world w iff i refers to w, namely:

$$M, g, w \models i \Leftrightarrow g(i) = w$$

In order to use nominals, Hybrid Logics introduce a new operator @ (read "at"). Given a nominal i and a formula A, $@_i A$ means that formula A holds at the world referenced by i. More precisely we have:

$$M, g, w \models @_i A \Leftrightarrow M, g, g(i) \models A$$

Interestingly, the @ operator defines a notion of semantical equality between nominals. In fact by using the rules given above, we have that

$$M, g, w \models @_i j \Leftrightarrow M, g, g(i) \models j \Leftrightarrow g(i) = g(j)$$

i.e., $@_i j$ is true iff both i and j refer to the same world.

In order to have further expressive power, it is possible to introduce a first order quantification over nominals. It is therefore possible to write formulas of the kind $\forall i A$, whose informal meaning is that A is true with regard to all the possible ways to assign a world to i, i.e.

$$M, g, w \models \forall i A \Leftrightarrow \forall u \in W, M, g[i \mapsto u], w \models A$$

where $g' = g[i \mapsto u]$ is the new function that coincides with g but for i, where $g'(i)$ is equal to u (note that this is the standard way to evaluate first order quantification with respect to a domain of values).

We conclude here with two very simple examples:

1. the formula $\forall i(@_i A \rightarrow \Box B)$
   has the informal meaning *"each world satisfying A has all the accessible worlds satisfying B".*

2. the formula $\exists i(@_i A) \rightarrow \exists j, k(\neg @_j k \wedge (@_j B \wedge @_k C))$
   has the informal meaning *"if there is a world satisfying A, then there are two distinct worlds, one satisfying B and the other one C, respectively".*

The paper contains a lot of quite complex examples dealing with @ and $\forall$ in the context of formulas specifying different kinds of constraints for DTD-based XML documents.