

# Semantic Verification of Web System Contents<sup>\*</sup>

Maria Alpuente<sup>1</sup> Michele Baggi<sup>2</sup> Demis Ballis<sup>3</sup> Moreno Falaschi<sup>2</sup>

<sup>1</sup> DSIC, Universidad Politécnic de Valencia  
Camino de Vera s/n, Apdo. 22012, 46071 Valencia, Spain.  
alpuente@dsic.upv.es

<sup>2</sup> Dip. di Scienze Matematiche e Informatiche  
Pian dei Mantellini 44, 53100 Siena, Italy.  
{baggi,moreno.falaschi}@unisi.it

<sup>3</sup> Dip. Matematica e Informatica  
Via delle Scienze 206, 33100 Udine, Italy.  
demis@dimi.uniud.it

**Abstract.** In this paper, we present a rule-based specification language to define and automatically check semantic as well as syntactic constraints over the informative content of a Web system. The language is inspired by the GVERDI language and significantly extends it by integrating ontology reasoning into the specification rules and by adding new syntactic constructs. The resulting language increases the expressiveness of the original one and enables a more sophisticated treatment of the semantic information related to the contents of the Web system.

## 1 Introduction

Web systems are very often *collaborative* applications in which many users freely contribute to update their contents (e.g. wikis, blogs, social networks, . . .). In this scenario, the task of keeping data correct and complete is particularly arduous, because of the very poor control over the content update operations which may easily lead to data inconsistency problems.

In this paper, we propose a rule-based specification language which allows one to formalize and automatically check semantic as well as syntactic properties over the *static* contents of *any* Web system. The language is inspired by the GVerdi specification language [2,4] and extends it in the following ways.

(i) Web contents (typically, XML/XHTML data) are frequently coupled with ontologies with the aim of equipping data with semantic information. Our specification language provides ontology reasoning capabilities which allow us to query a (possibly) remote ontology reasoner to check semantic properties over the data of interest, and to retrieve semantic information which may be combined with the syntactic one for improving the analysis.

---

<sup>\*</sup> This work has been partially supported by the EU (FEDER) and the Spanish MEC under grant TIN2007-68093-C02-02, Integrated Action HA 2006-0007, UPV-VIDI grant 3249 PAID0607, and the Italian MUR under grant RBIN04M8S8, FIRB project, Internationalization 2004.

(ii) We extend the GVerdi specification language with new rule constructs for the definition of conjunctions and disjunctions of patterns which can be recognized inside XML documents. The new constructs increase the expressiveness of the original language, since they enable the specification of a larger set of semantic as well as syntactic constraints.

(iii) Along with the specification language, we formulate a novel verification methodology which automatically checks a specification against the considered Web contents and discovers incorrect as well as incomplete information.

**Related work.** In recent years, several rule-based methodologies for validating the content of Web systems have been developed. In [6] constraint logic programming is applied to constrain the static content and the structure of a Web site, while [9,7] define type system and type checking techniques which are basically natural generalizations of DTDs and XML Schemas for describing and validating the structure of XML documents. [8] proposes a formal framework which uses inference rules and axioms to define some semantic properties concerning the content of Web sites. [The framework xlinkit \[11,10\] allows one to check the consistency of distributed, heterogeneous documents as well as to fix the \(possibly\) inconsistent information.](#) All the mentioned approaches, albeit very useful in their specific domains, share the same limitation: the syntactic as well as semantic constraints they specify only rely on the data to be checked. Our approach tries to overcome such limitation, on one hand, by introducing external sources of information (i.e. ontologies), which enrich the Web system content with additional semantic data; on the other hand, by enabling ontology reasoning for refining the verification process.

## 2 Description logics and ontologies

Description Logics (DLs) are logic formalisms for representing knowledge of application domains and reasoning about it. In particular, they [are suited means to model the semantics of XML data by defining ontologies endowing the raw XML data with semantic information.](#)

Ontologies are complex data structures which basically consist of concepts (classes), individuals (instances of classes) and roles (relations). An ontology can be queried to retrieve semantic data or to verify semantic properties related to a given XML repository.

In the following, we present a description logic language for querying ontologies, which extends the well-known description logic formalized within the OWL-DL[13] framework by (i) admitting the use of variables as placeholders for atomic concepts, roles and individuals; (ii) letting function calls compute atomic concepts, roles and individuals. Due to lack of space, we present a DL language without the description of the DL constructs. For a thorough discussion about them, please see [3].

Let  $\mathcal{A}$  be the set of all atomic concepts,  $\mathcal{R}$  be the set of all atomic roles,  $\mathcal{I}$  be the set of all individual names. Concepts are formalized via the following

abstract syntax:

$$C \rightarrow A \mid X \mid f(t_1, \dots, t_n) \mid \top \mid \perp \mid \neg C \mid C \sqcap C \mid C \sqcup C \mid \forall R.C \mid \exists R.C \mid \geq_n R \mid \leq_n R$$

where  $C$  represents a (complex) concept,  $A$  an atomic concept,  $R$  a role,  $n$  a natural number,  $X$  a concept variable,  $f(t_1, \dots, t_n)$  a call to the function  $f: \mathcal{D} \mapsto \mathcal{A}$  that maps elements of a given domain  $\mathcal{D}$  into the set of atomic concepts  $\mathcal{A}$ .

Roles can be described using the following abstract syntax:

$$R \rightarrow S \mid X \mid f(t_1, \dots, t_n)$$

where  $R$  represents a role,  $X$  a role variable,  $S$  an atomic role,  $f(t_1, \dots, t_n)$  a call to the function  $f: \mathcal{D} \mapsto \mathcal{R}$  that maps elements of a given domain  $\mathcal{D}$  into the set of atomic roles  $\mathcal{R}$ .

Finally, individuals are formalized via the following abstract syntax:

$$I \rightarrow a \mid X \mid f(t_1, \dots, t_n)$$

where  $a$  represents an individual name,  $X$  an individual variable,  $f(t_1, \dots, t_n)$  a call to the function  $f: \mathcal{D} \mapsto \mathcal{I}$  that maps elements of a given domain  $\mathcal{D}$  into the set of individual names  $\mathcal{I}$ .

Concepts, roles and individuals, which are built by applying the grammar rules given above, can be used to query ontologies using the standard query constructs of Table 1. Therefore, given an ontology  $ont$  and a query construct  $cst$ , an *ontology query* on  $ont$  is an expression of the form:  $ont.cst(a_1, \dots, a_n)$ , where  $a_i, i = 1, \dots, n$ , are the arguments of the query construct  $cst$ . An ontology query whose execution returns a boolean value (respectively, a set of values) is called *boolean* (respectively, *non-boolean*) ontology query.

Query construct	Description
$allConcepts()$	All concepts defined in the ontology
$allRoles()$	All roles defined in the ontology
$allIndividuals()$	All individuals defined in the ontology
$satisfiable(C)$	Is $C$ satisfiable?
$subsumes(C_1, C_2)$	Does $C_2 \sqsubseteq C_1$ ?
$disjoint(C_1, C_2)$	Does $C_1 \sqcap C_2 \equiv \emptyset$ ?
$children(C)$	All concepts which are children of $C$
$equivalents(C)$	All concepts which are equivalent to $C$
$instances(C)$	All individuals which belong to $C$
$instanceOf(a, C)$	Does $a$ belong to $C$ ?
$roleFillers(a, R)$	All individuals $b$ such that $R(b, a)$ holds
$related(R)$	All pairs $(a, b)$ such that $R(a, b)$ holds

**Table 1.** Ontology query constructs

*Example 1.* Let `++` denote the string concatenation operator. Assume an ontology `univ` modeling an academic domain is given, the boolean ontology query

$$\text{univ.instanceOf}(X++Y, \text{professor} \sqcap \text{female})$$

would check whether the individual, which is computed by concatenating the values associated with variables `X` and `Y`, is a female professor (i.e. an instance of the concept `professor`  $\sqcap$  `female`).

**Ontology query execution.** Typically, ontology queries are sent to an ontology reasoner and then executed against an ontology of interest. Standard ontology reasoners are able to execute only ground, flat ontology queries, that is, queries without variables and function calls. Note that our language allows one to generate ontology queries containing variables as well as function calls. Thus, in order to execute such queries using a standard reasoner, we need to made the query ground and to evaluate all the functions calls before sending the query to the reasoner. This process of query manipulation will be made explicit in the following sections.

### 3 The Web specification language

Our specification language allows us to formalize and verify properties over the content of a Web system.

**Web content denotation.** Throughout this paper, we assume that the data to be checked are stored into an XML repository. Moreover, since XML data are provided with a tree-like structure, we model XML repositories as finite sets of *ground* terms (that is, terms not containing variables) of a suitable term algebra. In the following, we will also consider *Web templates*, which are terms of a non-ground term algebra, which may contain variables. Web templates are used for specifying patterns to be recognized in XML repositories. See [2] for more details.

**Web specifications.** The language provides constructs for specifying two kinds of rules: *correctness* rules and *completeness* rules. The former describe constraints for detecting erroneous information into a given XML repository, while the latter recognize incomplete/missing information. Both kinds of rules may be *conditional*, that is, they can be fired if and only if an associated condition holds.

A *condition* is a finite (possibly *empty*) sequence  $c_1, \dots, c_n$ , where each  $c_i$  may be (i) a membership test w.r.t. a regular language of the form  $X \in \text{rexp}^4$ , (ii) an equation  $s = t$ , where  $s$  and  $t$  are expressions which may contain nested function calls to be evaluated<sup>5</sup> w.r.t. a given term rewriting system (TRS)  $R$ , and (iii) a boolean ontology query.

Given a substitution  $\sigma$ , which takes an expression and replaces its variables with ground terms and a condition  $C \equiv c_1, \dots, c_n$ , we say that  $C$  *holds* for  $\sigma$  iff each  $c_i\sigma$  is ground and

<sup>4</sup> Regular languages are denoted by the usual Unix-like regular expression syntax.

<sup>5</sup> In our framework, equation evaluation is handled by standard rewriting [12].

- if  $c_i \equiv X \in \mathbf{rexp}$ , then  $X\sigma \in \mathcal{L}(\mathbf{rexp})$ , where  $\mathcal{L}(\mathbf{rexp})$  is the regular language described by  $\mathbf{rexp}$ ;
- if  $c_i \equiv (s = t)$ , then **the equality  $(s = t)\sigma$  holds in a given TRS  $R$** .
- if  $c_i \equiv B$ , with  $B$  boolean ontology query, then the execution of  $B\sigma$  returns *true*<sup>6</sup>.

Now, we are ready to introduce correction as well as completeness rules.

**Definition 1 (Correctness rule).** A correctness rule is an expression of the form

$$\bigwedge_{i=1}^n \mathbf{l}_i \rightarrow \mathbf{error} \mid \mathbf{C}$$

where each  $\mathbf{l}_i$  is a Web template, **error** is a reserved constant,  $\mathbf{C}$  is a condition.

Informally, the meaning of a correctness rule  $\mathbf{l}_1 \wedge \dots \wedge \mathbf{l}_n \rightarrow \mathbf{error} \mid \mathbf{C}$  is as follows. Whenever an instance  $\mathbf{l}_i\sigma$  of  $\mathbf{l}_i$  for each  $i \in \{1, \dots, n\}$  is recognized in some Web page  $\mathbf{p}$ , and the rule condition  $\mathbf{C}$  holds for  $\sigma$ , then Web page  $\mathbf{p}$  is signaled as an incorrect page.

*Example 2.* Consider an XML repository containing academic information along with an ontology `univ` modeling such a domain. Suppose we want to verify the following property: *if an associate professor has more than three Ph.D. students, then he cannot be titular of more than one course*. Then a possible correctness rule formalizing such a property might be

```
use './Ontologies/UniversityDomain' as univ
course(cId(X), titular(name(Y)))
^ course(cId(Z), titular(name(Y))) → error |
  univ.instanceOf(Y, AssocProf □ (≥3hasStd □ ∀hasStd.PhDStudent)),
  X ≠ Z
```

To define completeness rules, we need the following auxiliary notion. Given an expression  $e$ , by  $Var(e)$  we denote the set of all the variable appearing in  $e$ .

**Definition 2 (Completeness rule).** A completeness rule is an expression of the form

$$\bigwedge_{i=1}^n \mathbf{l}_i \rightarrow \bigvee_{j=1}^m \mathbf{r}_j \mid \mathbf{C} \text{ containing ct } \langle \mathbf{q} \rangle$$

where each  $\mathbf{l}_i, \mathbf{r}_j$  are Web templates,  $\mathbf{C}$  is a condition, **containing ct** is optional, where **ct** is a ground term,  $\mathbf{q} \in \{\mathbf{A}, \mathbf{E}\}$ , and  $\bigcup_{j=1}^m Var(\mathbf{r}_j) \cup Var(\mathbf{C}) \subseteq Var(\mathbf{l})$ .

Completeness rule are called *universal* (resp., *existential*), whenever  $\mathbf{q} \equiv \mathbf{A}$  (resp.,  $\mathbf{q} \equiv \mathbf{E}$ ).

Intuitively, given an XML repository  $W$ , the interpretation of a universal (resp., existential) rule  $\mathbf{l}_1 \wedge \dots \wedge \mathbf{l}_n \rightarrow \mathbf{r}_1 \vee \dots \vee \mathbf{r}_m \mid \text{containing ct } \langle \mathbf{A} \rangle$  (resp.,  $\langle \mathbf{E} \rangle$ ) w.r.t.  $W$  is as follows: if an instance  $\mathbf{l}_i\sigma$  of  $\mathbf{l}_i$  for each  $i \in \{1, \dots, n\}$  is

<sup>6</sup> We assume that all the function calls appearing in  $B\sigma$  are evaluated before executing the query.

recognized in some  $p \in W$  and the condition  $C$  holds for  $\sigma$ , then an instance  $r_j\sigma$  of at least one  $r_j$ ,  $j \in \{1, \dots, m\}$  must be recognized in *all* (resp. *some*) XML documents of  $W$  containing the  $ct$  term. Roughly speaking,  $ct$  provides the “scope” of the quantification and allows us to compute the part of the XML repository which is checked by the rule; if  $ct$  is not specified the rule is applied to the whole repository.

*Example 3.* Consider again an academic XML repository along with the usual ontology `univ`. We want to verify that *for each course, given by a full professor, at least two exam dates must be provided*. A completeness rule formalizing this property might be

```
use './Ontologies/UniversityDomain' as univ
course(cId(X))  $\rightarrow$  course(cId(X),examDate(),examDate()) |
                        univ.instanceOf(X,  $\exists$ CourseGivenBy.FullProf)  $\langle E \rangle$ 
```

Finally, we define a *Web specification* as a **triple**  $(I_N, I_M, R)$ , where  $I_N$  is a set of correctness rules,  $I_M$  is a set of completeness rules and  $R$  is a term rewriting system. Given an XML repository and a Web specification, diagnoses are carried out by running the Web specification rules against the XML repository.

**Web specifications with meta-symbols.** Sometimes, it is particularly fruitful to consider rules containing Web templates which may subsume several meanings. To this purpose, completeness and correctness rules may include special *meta-symbols* into the Web templates which are associated with non-boolean ontology queries.

Web specification rules containing meta-symbols have to be pre-processed before being executed on a given XML repository.

The following definition is auxiliary. Let  $e$  be a (syntactic) expression of our language,  $m$  be a meta-symbol,  $v$  be a symbol. By  $e[m/v]$  we denote the expression  $e'$  obtained from  $e$  by replacing each occurrence of  $m$  with  $v$ .

Basically, we expand each rule  $r$  containing meta-symbols as follows:

- for each meta-symbol  $m$  appearing in  $r$ , we execute the associated ontology query and we collect the results  $\{v_1, \dots, v_n\}$ ;
- if  $m$  appears in the left-hand side of  $r$ , we replace  $r$  with the rules  $r_1[m/v_1], \dots, r_n[m/v_n]$ .
- if  $m$  appears in a disjunct  $\rho$  of the right-hand side of  $r$ , we replace  $\rho$  in  $r$  with  $\rho[m/v_1] \vee \dots \vee \rho[m/v_n]$ .

A full description of the expansion algorithm is available in [1].

For the sake of clarity, let us see an example.

*Example 4.* Consider again an academic XML repository along with the usual ontology `univ`. We want to specify that *email or post address have to be specified for each university professor*. Assume that the ontology `univ` contains (i) the concept `contactInfo` whose subconcepts are `email` and `address`; and (ii) the concept `Professor` whose subconcepts are `AssociateProf` and `FullProf`.

We might model the considered property using a universal completeness rule containing two meta-symbols (namely, `contact` and `prof`).

```

use './Ontology/UniversityDomain' as univ
metasymbol contact: univ.getChildren("contactInfo")
metasymbol prof: univ.getChildren("Professor")
prof(name(X))  $\rightarrow$  prof(name(X),contact()) | containing member() <A>

```

By expanding the considered completeness rule, we generate the following set of rules without meta-symbols.

```

use './Ontology/UniversityDomain' as univ
AssociateProf(name(X))  $\rightarrow$  AssociateProf(name(X),email())  $\vee$ 
                          AssociateProf(name(X),address()) |
                          containing member() <A>
FullProf(name(X))  $\rightarrow$  FullProf(name(X),email())  $\vee$ 
                      FullProf(name(X),address()) |
                      containing member() <A>

```

## 4 Verification Methodology

In this section we present a methodology to automatically verify a given XML repository w.r.t. a Web specification. Without loss of generality, we only consider Web specifications without meta-symbols, since any Web specification with meta-symbols can be transformed into an equivalent one without meta-symbols as explained in Section 3.

We proceed as follows: first we describe the *partial rewriting* [2] mechanism which allows us to detect patterns inside XML documents and rewrite them. Then, we will employ this evaluation mechanism to check correctness and completeness of an XML repository.

### 4.1 Simulation and partial rewriting.

Simulation allows us to recognize the structure and the labeling of a given Web template into a particular XML document. It can be formally defined as follows.

**Definition 3.** *The simulation relation  $\trianglelefteq$  on terms is the least relation satisfying the rule  $f(\mathbf{t}_1, \dots, \mathbf{t}_m) \trianglelefteq g(\mathbf{s}_1, \dots, \mathbf{s}_n)$  iff  $f \equiv g$  and  $\mathbf{t}_i \trianglelefteq \mathbf{s}_{\pi(i)}$ , for  $i = 1, \dots, m$ , and some injective function  $\pi : \{1, \dots, m\} \rightarrow \{1, \dots, n\}$ .*

W.l.o.g., we disregard quantifiers from Web specification rules.

**Definition 4 (Partial rewriting).** *Let  $\mathbf{s}, \mathbf{t}$  be terms and **or**, **error** two fresh constructor symbols. We denote by  $\mathbf{s}|_e$  the subterm of  $\mathbf{s}$  rooted at position  $e$ . We say that  $\mathbf{s}$  partially rewrites to  $\mathbf{t}$  via rule  $r \equiv \bigwedge_{i=1}^n \mathbf{l}_i \rightarrow \mathbf{Rhs} \mid C$  and substitution  $\sigma$  (in symbols  $\mathbf{s} \rightarrow_r \mathbf{t}$ ) if and only if there exist positions  $u_1, \dots, u_n$  in  $\mathbf{s}$  such that*

- (i)  $\mathbf{l}_i \sigma \trianglelefteq \mathbf{s}|_{u_i}$  for all  $i \in \{1, \dots, n\}$ ;

- (ii)  $C$  holds for  $\sigma$ ;
- (iii) if  $\text{Rhs} \equiv \bigvee_{j=1}^m \mathbf{r}_j$  then  $\mathbf{t} \equiv \text{or}(\mathbf{r}_1\sigma, \dots, \mathbf{r}_m\sigma)$
- (iv) if  $\text{Rhs} \equiv \text{error}$  then  $\mathbf{t} = \text{error}(\mathbf{s}, u_1, \dots, u_n)$

## 4.2 Detecting correctness errors.

Given a Web specification  $(I_N, I_M, R)$ , to detect erroneous or undesirable data included in an XML repository  $W$ , we have to execute all the correctness rule in  $I_N$  against each XML document  $p$  belonging to  $W$ . The procedure is described in the following definition.

**Definition 5.** Let  $W$  be an XML repository and  $(I_N, I_M, R)$  be a Web specification. Given  $\mathbf{p} \in W$ , we say that  $\mathbf{p}$  is incorrect w.r.t.  $(I_N, I_M, R)$ , if there exists a correctness rule  $r \equiv (\bigwedge_{i=1}^n \mathbf{l}_i \rightarrow \text{error} \mid \mathbf{C}) \in I_N$  such that

- (i)  $\mathbf{p}$  partial rewrites to  $\text{error}(p, u_1, \dots, u_n)$  via  $r$  and substitution  $\sigma$ ;
- (ii)  $\mathbf{C}$  holds for  $\sigma$ . We say that  $(\bigwedge_{i=1}^n \mathbf{l}_i)\sigma$  is an incorrectness symptom for  $\mathbf{p}$ .

Note that the generated term  $\text{error}(p, u_1, \dots, u_n)$  provides all the needed information to precisely locate the incorrectness symptom inside  $p$ .

## 4.3 Detecting completeness errors.

The verification of an XML repository w.r.t. a set of completeness rules of a Web specification needs a more complex analysis. In the following we introduce some semantic foundations we require to formalize the analysis.

**Completeness rule semantics.** A *completeness requirement* (or simply *requirement*) is a triple  $\langle \mathbf{e}, \mathbf{q}, \mathbf{ct} \rangle$ , where  $\mathbf{e}$  and  $\mathbf{ct}$  are ground terms and  $\mathbf{q} \in \{\mathbf{A}, \mathbf{E}\}$ . A requirement is called *universal* whenever  $\mathbf{q} = \mathbf{A}$ , while it is called *existential* whenever  $\mathbf{q} = \mathbf{E}$ . Sometimes the components  $\mathbf{q}, \mathbf{ct}$  of a requirement can be left undefined, in this case we simply omit them and write  $\langle \mathbf{e}, -, - \rangle$ . This kind of requirements are called *initial* requirements.

Let  $\langle \mathbf{e}, \mathbf{q}, \mathbf{ct} \rangle$  be a requirement,  $r \equiv \text{Lhs} \rightarrow \bigvee_{j=1}^m \mathbf{r}_j \mid \mathbf{C}$  containing  $\mathbf{ct}_r \langle \mathbf{q}_r \rangle \in I_M$  be a rule such that  $s \equiv \mathbf{e} \rightarrow_r \text{or}(\mathbf{h}_1, \dots, \mathbf{h}_m)$ . We define the tree  $T_s$  associated with the partial rewriting step  $s$  as  $\text{or}(\langle \mathbf{h}_1, \mathbf{q}_r, \mathbf{ct}_r \rangle, \dots, \langle \mathbf{h}_m, \mathbf{q}_r, \mathbf{ct}_r \rangle)$ .

**Definition 6 (Production step).** Let  $(I_N, I_M, R)$  be a Web specification and  $\mathbf{re} \equiv \langle \mathbf{e}, \mathbf{q}, \mathbf{ct} \rangle$  be a requirement. Let  $s_1, \dots, s_k$  be all partial rewriting steps which rewrites  $\mathbf{e}$  using the rules in  $I_M$ . Let  $T_{s_1}, \dots, T_{s_k}$  be the trees associated with the partial rewriting steps  $s_1, \dots, s_k$ . The production step on  $\mathbf{re}$  w.r.t.  $I_M$  builds the tree  $\mathbf{re}(T_{s_1}, \dots, T_{s_k})$ .

Note that, if there is no rule  $r \in I_M$  such that  $\mathbf{e} \rightarrow_r \mathbf{t}$ , we say that  $\mathbf{re} \equiv \langle \mathbf{e}, \mathbf{q}, \mathbf{ct} \rangle$  is *irreducible*.

Let us now use the production step to define the finite maximal derivation tree for a requirement.

**Definition 7 (Derivation tree).** *Given a requirement  $\mathbf{re}$  and a Web specification  $(I_N, I_M, R)$ , a derivation tree for  $\mathbf{re}$  w.r.t. the set  $I_M$ , is defined as follows:*

- $\mathbf{re}$  is a derivation tree for  $\mathbf{re}$  w.r.t. the set  $I_M$ ;
- if  $T$  is a derivation tree for  $\mathbf{re}$  w.r.t. the set  $I_M$  and  $\mathbf{re}'$  is a requirement labeling a leaf of  $T$ , then the tree  $T'$  obtained from  $T$  by replacing  $\mathbf{re}'$  with the tree generated by applying a production step on  $\mathbf{re}'$  w.r.t.  $I_M$ , is a derivation tree for  $\mathbf{re}$  w.r.t. the set  $I_M$ .

A maximal derivation tree  $T_{\mathbf{re}}$  for  $\mathbf{re}$  w.r.t.  $I_M$  is a derivation tree where all leaves are labeled with an irreducible requirement and the set of requirements labeling  $T_{\mathbf{re}}$  nodes is finite.

It follows that the maximal derivation tree for a requirement  $\mathbf{re}$  w.r.t.  $I_M$  contains all the requirements that can be derived from  $\mathbf{re}$  w.r.t.  $I_M$ . Since the derivable requirements from  $\mathbf{re}$  w.r.t.  $I_M$  could be infinite, a maximal derivation tree for  $\mathbf{re}$  might not exist. In [1], we propose some syntactical restrictions over Web specifications, to ensure that the set of derivable requirements for a requirement  $\mathbf{re}$  w.r.t. a set of completeness rules is finite and hence a maximal derivation tree for  $\mathbf{re}$  exists. Moreover, we provide a way to obtain from a maximal derivation tree, an equivalent finite structure.

**Diagnoses of completeness errors.** Let  $(I_N, I_M, R)$  be a Web specification,  $W$  be an XML repository. An XML document  $p \in W$  can be associated with the initial requirement  $\mathbf{re}_p = \langle \mathbf{p}, -, - \rangle$ . Thus, we can compute the maximal derivation tree for  $\mathbf{re}_p$  w.r.t.  $I_M$ , which contains *all* the requirements that can be derived from  $p$  using the rules in  $I_M$  by partial rewriting.

Now, to diagnose completeness errors in  $W$ , we can proceed as follows. For each  $p \in W$ , (i) we compute the maximal derivation tree  $T_p$  for  $\langle \mathbf{p}, -, - \rangle$  w.r.t.  $I_M$ , then (ii) we traverse  $T_p$  and we check whether the requirements occurring in  $T_p$  are not satisfied. The verification process terminates delivering the detected completeness errors.

**Definition 8 (Requirement unsatisfiability).** *Let  $W$  be an XML repository,  $(I_N, I_M, R)$  be a Web specification and  $\mathbf{re} \equiv \langle \mathbf{e}, \mathbf{q}, \mathbf{ct} \rangle$  be a requirement. Let  $\text{TEST}_{\mathbf{re}} = \{p \in W \mid \mathbf{ct} \leq p\}$ .*

- If  $\mathbf{re}$  is a universal requirement, then  $\mathbf{re}$  is not satisfied in  $W$  if one of the following conditions hold:
  1.  $\text{TEST}_{\mathbf{re}} = \emptyset$ ;
  2. there exists  $\mathbf{p} \in \text{TEST}_{\mathbf{re}}$  s.t.  $\mathbf{e} \not\leq \mathbf{p}$ .
- If  $\mathbf{re}$  is an existential requirement, then  $\mathbf{re}$  is not satisfied in  $W$  if for each  $\mathbf{p} \in \text{TEST}_{\mathbf{re}}$ ,  $\mathbf{e} \not\leq \mathbf{p}$ .

Dually, we can define universal/existential requirement satisfiability.

By using the previous definitions, we formalize the following completeness analysis.

**Definition 9 (Maximal derivation tree analysis).** *Let  $W$  be an XML repository,  $p \in W$ ,  $(I_N, I_M, R)$  be a Web specification, and  $T_p$  be the maximal derivation tree of  $\langle p, -, - \rangle$  w.r.t.  $I_M$ . The completeness analysis of a maximal derivation tree is inductively defined on the structure of  $T_p$  by the following function:*

$$\text{Verify}(\text{or}(T_1, \dots, T_k)) = \begin{cases} \text{error}(e_1, \dots, e_k) & \text{if } \forall i T_i \text{ is not satisfied in } W \\ \text{Verify}(T_i) \forall i \text{ s.t. } T_i \text{ is satisfied in } W & \text{otherwise} \end{cases}$$

where  $\text{root}(T_i) = \langle e_i, q_i, ct_i \rangle$ .

$$\text{Verify}(\text{re}(T_1, \dots, T_k)) = \text{Verify}(T_i) \forall i = 1, \dots, k$$

The term  $\text{error}(e_1, \dots, e_k)$  represents a completeness error which means that no disjunct  $e_i$  for  $i \in \{1, \dots, k\}$  is recognized into the considered document. Roughly speaking, the execution of  $\text{Verify}(T_p)$  finds all the completeness errors inside an XML document  $p$  w.r.t.  $I_M$ . By applying this verification methodology to all XML documents in  $W$ , we can perform the completeness analysis of the whole XML repository  $W$ .

## 5 Conclusions

In this paper we presented a rule-based specification language, inspired by GVerdi [2,4], for formalizing and automatically checking semantic and syntactic properties over Web system contents. We increased the GVerdi language expressiveness by adding new constructs for defining conjunctive as well as disjunctive Web patterns. Moreover, we exploited ontology reasoning capabilities to retrieve semantic information which is useful to refine the verification process. In our framework, the integration between the language and the ontology reasoner is achieved by means of a description logic which extends the well-known OWL-DL[13] framework by admitting variables and function calls among the usual DL constructs. We are currently developing a prototypical system of our framework which implements ontology reasoning via the DIG Interface [5], which is an XML API for connecting applications to description logic reasoners.

## References

1. Alpuente, M., Baggi, M., Ballis, D., Falaschi, M.: Semantic Verification of Web System Contents. Technical Report, University of Udine (2008)
2. Alpuente, M., Ballis, D., Falaschi, M.: Automated Verification of Web Sites Using Partial Rewriting. *Software Tools for Technology Transfer* 8, 565–585 (2006)
3. Calvanese, D., McGuinness, D., Nardi, D., Patel-Scheider, P., Baader, F.: *The Description Logic Handbook*. Cambridge University Press (2003)
4. Ballis, D., García Vivó, J.: A Rule-based System for Web Site Verification. In: *WWW 2005. ENTCS*, vol. 157(2), Elsevier (2005)
5. Bechhofer, S.: *The DIG Description Logic Interface: DIG/1.1*. Technical report, University of Manchester (2003)
6. Florido, M., Coelho, J.: VeriFlog: Constraint Logic Programming Applied to Verification of Website Content. In: *APWEB 2006, LNCS*, vol. 3842, pp. 148–156. Springer, Heidelberg (2006)

7. Florido, M., Coelho, J.: Type-based Static and Dynamic Website Verification. In: ICIW 2007. IEEE Computer Society Press (2007)
8. Trousse, B., Despeyroux, T.: Semantic Verification of Web Sites Using Natural Semantics. In: RIAO 2000 (2000)
9. Pierce, B., Hosoya, H.: Regular Expressions Pattern Matching for XML. In: POPL 2001, ACM SIGPLAN Notices 36(3), pp. 67–80 (2001)
10. Nentwich, C., Emmerich, W., Finkelstein, A.: Consistency Management with Repair Actions. In: ICSE 2003, pp. 455–464. IEEE Computer Society (2003)
11. Ellmer, E., Emmerich, W., Finkelstein, A., Nentwich, C.: Flexible Consistency Checking. ACM Transaction on Software Engineering 12(1), pp. 28–63 (2003)
12. J.W. Klop.: Term Rewriting Systems. In: S. Abramsky, D. Gabbay, and T. Maibaum (eds.): Handbook of Logic in Computer Science. vol. I, pp. 1–112, Oxford University Press (1992)
13. World Wide Web Consortium (W3C). OWL Web Ontology Language Guide (2004). <http://www.w3.org/>