Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Electronic Notes in Theoretical Computer Science 124 (2005) 41–61
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

A Rewriting-based Framework for Web Sites Verification [1]

M. Alpuente

DSIC, Universidad Politécnica de Valencia, Camino de Vera s/n, Apdo. 22012, 46071 Valencia, Spain. Email: alpuente@dsic.upv.es.

D. Ballis M. Falaschi

Dip. Matematica e Informatica, Via delle Scienze 206, 33100 Udine, Italy. Email: {demis,falaschi}@dimi.uniud.it.

Abstract

In this paper, we develop a framework for the automated verification of Web sites which can be used to specify integrity conditions for a given Web site, and then automatically check whether these conditions are fulfilled. First, we provide a rewriting-based, formal specification language which allows us to define syntactic as well as semantic properties of the Web site. Then, we formalize a verification technique which obtains the requirements not fulfilled by the Web site, and helps to repair the errors by finding out incomplete information and/or missing pages. Our methodology is based on a novel rewriting-based technique, called *partial rewriting*, in which the traditional pattern matching mechanism is replaced by tree *simulation*, a suitable technique for recognizing patterns inside semistructured documents. The framework has been implemented in the prototype Web verification system VERDI which is publicly available.

Keywords: Formal Web site verification, specification languages, rewriting, simulation.

1 Introduction

The increasing complexity of Web sites has turned their design and construction into a challenging problem. Systematic, formal approaches can bring

[1] This work has been partially supported by MCYT under grants TIC2001-2705-C03-01,HU2003-0003 and by Generalitat Valenciana under grant GR03/025, and by ICT for EU-India Cross Cultural Dissemination Project under grant ALA/95/23/2003/077-054.

doi:10.1016/j.entcs.2004.07.014

many benefits to quality Web site construction, giving support for automated Web site verification. This paper presents an approach to Web site specification and verification based on rewriting-like machinery. We use rewriting-based technology both to specify the integrity conditions and to formalize a verification technique which obtains the requirements not fulfilled by the Web site, and then is able to repair errors by finding out missing pages and/or incomplete information, such as the data or the links available in a particular page.

Although the management of Web sites has received significant attention in recent years [6,12,13], few works address the semantic verification of Web sites. [16] presents *regular expression types*, which are natural generalizations of DTDs describing structures in XML documents. In this framework, the problem of verifying the structure of an XML document boils down to a type-checking problem; that is, an XML document is well-structured, if it is well-typed. In [13], a declarative verification algorithm is proposed which checks a particular class of integrity constraints concerning the Web site's structure, but not the contents of a given instance of the site. [12] proposes a methodology which consists of using inference rules and axioms to define some semantic constraints concerning the Web site contents. Then, a verification technique is proposed which is based on compiling the specification into Prolog code. Our idea in this paper is that term rewriting techniques can support in a natural way not only intuitive, high level Web site specification, but also efficient Web site verification and repairing techniques. As far as we know, rewriting-based techniques have not been explored in this context to date. We only know of two related approaches which focus on transformation rather than verification issues: a rewriting-based implementation is provided in [17] for (a fragment of) XSLT, the rule-based language designed by W3C for the transformation of XML documents, whereas rewrite rules are used in [3] to perform HTML transformations with the aim of improving Web applications by cleaning up syntax, reorganizing frames, or updating to new standards.

Our contribution. We first provide a rewriting-based, formal specification language which allows us to define conditions on both the structure and the contents of Web sites in a simple and concise way. For instance, it allows us to enforce that some information is available at a given Web page, some links between pages do exist or even the existence of the Web pages themselves. In our formalism, web pages (HTML/XML documents) are modeled as Herbrand terms, and, consequently, Web sites are finite sets of terms. Then, we formalize a verification technique in which a Web site is checked w.r.t. a given Web specification in order to detect incomplete and/or missing Web pages. Moreover, by analyzing the requirements not fulfilled by the Web site, we are

also able to find out the missing information which is needed to repair the Web site. Since reasoning on the Web calls for formal methods specifically fitting the Web context, we develop a novel, rewriting-based technique called *partial rewriting*, in which the traditional pattern matching mechanism is replaced with tree *simulation* [15] in order to provide a suitable mechanism for recognizing patterns inside semistructured documents. The notion of simulation has been already used before for dealing with semistructured data in a number of query and transformation languages [6,8,14,10]. The reason is twofold: on the one hand, it provides a powerful method to extract information from semistructured data; on the other hand, efficient algorithms exist for computing simulations [15]. To assess the feasibility and efficiency of our approach, we have implemented the prototype system VERDI (VErification and Rewriting for Debugging Internet sites), which is based on the verification methodology that we propose and is publicly available online.

Plan of the paper. Section 2 summarizes some preliminary definitions and notations. In Section 3, we formulate a simple method for translating HTML/XML documents into Herbrand terms. Section 4 is devoted to formalize the specification language, whereas Section 5 formalizes the *partial rewriting* mechanism, which is based on page simulation. In Section 6, we introduce our verification technique, which is formalized as a fixpoint computation. First, the set of requirements to be fulfilled by the Web site `W` is computed as the fixpoint of a suitable operator associated with the Web site specification `I`. Then, by using simulations we select those requirements which are not satisfied by `W` and the corresponding incomplete/missing Web pages which are the source for the errors. The requirements which are not satisfied also allow us to ascertain the missing information which is needed to repair the Web site. Some notes regarding the implementation of the system VERDI are given in Section 7. Section 8 concludes. More details and missing proofs can be found in [2].

2 Preliminaries

We call *alphabet* a finite set of symbols. Given the alphabet A, A^* denotes the set of all finite sequences of elements over A. Syntactic equality between objects is represented by $\equiv$.

By $\mathcal{V}$ we denote a countably infinite set of variables and Σ denotes a set of function symbols, or *signature*. We consider varyadic signatures (i.e. signatures in which function symbols have an unbounded arity, that is, they may be followed by an arbitrary number of arguments) as in [11]. $\tau(\Sigma, \mathcal{V})$ and

$\tau(\Sigma)$ denote the *non-ground term algebra* and the *term algebra* built on $\Sigma \cup \mathcal{V}$ and Σ, respectively. $\tau(\Sigma)$ is usually called the Herbrand universe over Σ. A term t is *linear*, if no variable appears more than once in t.

Terms are viewed as labelled trees in the following way: a term in $\tau(\Sigma)$ is a tree $(V, E, r, \mathit{label})$, where V is a set of vertices, E is a set of edges (i.e. pairs of vertices), $r \in V$ is the *root* vertex and *label* is a *labeling* function such that $\mathit{label}(v) \in (\Sigma \cup \mathcal{V})$, for each $v \in V$. Let us see a small example.

Example 2.1 Consider the term $t \equiv (f(g(a), X))$ in $\tau(\{f, g, a\}, \{X\})$. Term t can be represented by the structure $(V, E, r, \mathit{label})$, where $V = \{v_0, v_1, v_2, v_3\}$, $E = \{(v_0, v_1), (v_0, v_2), (v_1, v_3)\}$, $r \equiv v_0$, and function *label* is defined as follows: $\mathit{label}(v_0) = f$, $\mathit{label}(v_1) = g$, $\mathit{label}(v_2) = X$, $\mathit{label}(v_3) = a$.

Given two vertices $v, v' \in V$ of a term $t \equiv (V, E, r, \mathit{label})$, by $v \geq v'$ we mean that v is a *descendant* of v' in t. By $t|_v$ we mean the subterm rooted at vertex v of t. We denote the *depth* of a vertex v in a term t, that is the number of edges between r and v in t, as $\mathit{depth}(t, v)$. A *substitution* $\sigma \equiv \{X_1/t_1, X_2/t_2, \ldots\}$ is a mapping from the set of variables $\mathcal{V}$ into the set of terms $\tau(\Sigma, \mathcal{V})$. By $\mathit{Var}(t)$ we denote the set of variables occurring in term t.

In the following, we consider marked terms. Given Σ and $\mathcal{V}$, we denote the *marked* version of Σ ($\mathcal{V}$, respectively) as $\underline{\Sigma}$ ($\underline{\mathcal{V}}$, respectively). A syntactic object $\underline{o} \in \underline{\Sigma} \cup \underline{\mathcal{V}}$ is called the *marked version* of $o \in \Sigma \cup \mathcal{V}$. Given a term $t \equiv (V, E, r, \mathit{label}) \in \tau(\Sigma, \mathcal{V})$, a *marking* for t is a (boolean) function $\mu\colon V \to \{\mathit{yes}, \mathit{no}\}$. The *empty* marking ε for t is a marking for t such that $\varepsilon(v) = \mathit{no}$, for each $v \in V$. We define the *marked part* of a term t as

$$\mathit{mark}(t, \mu) \equiv (\{v \in V \mid \mu(v) = \mathit{yes}\}, \{(v_1, v_2) \in E \mid \mu(v_1) = \mu(v_2) = \mathit{yes}\}, r, \mathit{label}\ ).$$

A *valid* marking μ for a term $t \equiv (V, E, r, \mathit{label})$ is the empty marking for t or a marking for t such that the two following conditions hold:

(i) $\mu(r) = \mathit{yes}$;

(ii) $\mathit{mark}(t, \mu)$ is a term in $\tau(\Sigma, \mathcal{V})$.

Given a term $t \equiv (V, E, r, \mathit{label})$ and a valid marking μ for t, by slightly abusing

notation we recursively define a *marked* term $\mu(t)$ as follows:

$$\mu(t) = \begin{cases} \underline{X} & t \equiv (\{v\}, \emptyset, v, label) \wedge label(v) = X \in \mathcal{V} \\ & \wedge\, \mu(v) = yes \\ X & t \equiv (\{v\}, \emptyset, v, label) \wedge label(v) = X \in \mathcal{V} \\ & \wedge\, \mu(v) = no \\ \underline{f}(\mu(t_1), \ldots, \mu(t_n)) & t \equiv (V, E, r, label) \equiv f(t_1, \ldots, t_n) \wedge\ \mu(r) = yes \\ f(\mu(t_1), \ldots, \mu(t_n)) & t \equiv (V, E, r, label) \equiv f(t_1, \ldots, t_n) \wedge\ \mu(r) = no \end{cases}$$

When no confusion can arise, we simply denote the marked term $\varepsilon(t)$ by t.

Example 2.2 Consider again term $t \equiv (f(g(a), X))$ of Example 2.1. Let μ_1 be a marking for t defined as $\mu_1(v_0) = \mu_1(v_2) = \mu_1(v_3) = yes$, $\mu_1(v_1) = no$. Additionally, let μ_2 be a marking for t such that $\mu_2(v_0) = \mu_2(v_1) = yes$, $\mu_2(v_2) = \mu_2(v_3) = no$. Note that μ_1 is not a valid marking for t as the marked part of t is not a term in $\tau(\{f, g, a\}, \{X\})$, whereas μ_2 is valid for t and $\mu_2(t) = \underline{f}(\underline{g}(a), X)$ is a marked term.

3 Denotation of Web Sites

In this paper, a *Web page* is either an XML[20] or an HTML[19] document, and a *Web site* is a finite collection of Web pages. In the sequel, we provide a formalization of these concepts by means of semistructured expressions, which can be seen as an abstract syntax which generalizes the two markup languages XML and HTML. Then, we show how semistructured expressions can be translated into ordinary terms of a given term algebra in such a way that Web sites are represented as finite sets of (ground) terms.

3.0.1 Semistructured Expressions.

XML/HTML documents consist of nested structured data, which can be defined inductively. Abstracting from XML and HTML, we give a formal definition of semistructured expressions which are suitable for representing structured documents written in one of these two languages.

Let us consider two alphabets T and $\mathcal{T}ag$. We denote the set T^* by $\mathcal{T}ext$. An object $\mathtt{t} \in \mathcal{T}ag$ is called *tag* element, while an element $\mathtt{w} \in \mathcal{T}ext$ is called *text* element. A *semistructured expression* $\mathtt{e}$ over $\mathcal{T}ext$ and $\mathcal{T}ag$ sets can be

specified by the following syntax [2]

$$\begin{array}{rcl} \mathtt{e} & := & \mathtt{<t>\ elist\ </t>} \ \mid \ \mathtt{w} \qquad \forall\, \mathtt{w} \in \mathcal{T}ext, \mathtt{t} \in \mathcal{T}ag \\ \mathtt{elist} & := & \mathtt{e\ elist} \ \mid \ \epsilon \end{array}$$

We denote the set of all the semistructured expressions over $\mathcal{T}ext$ and $\mathcal{T}ag$ by $\mathcal{S}(\mathcal{T}ext, \mathcal{T}ag)$. Note that $\mathcal{T}ext \subseteq \mathcal{S}(\mathcal{T}ext, \mathcal{T}ag)$.

Example 3.1 The following object is a semistructured expression.

```
<members>
    <member>
       <name> mario </name>
       <surname> rossi </surname>
       <status> professor </status>
    </member>
    <member>
       <name> franca </name>
       <surname> bianchi </surname>
       <status> technician </status>
    </member>
    <member>
       <name> giulio </name>
       <surname> verdi </surname>
       <status> student </status>
    </member>
</members>
```

Roughly speaking, a semistructured expression is either a raw or a structured piece of text, where the structure is provided by tags. Consequently, tags allow us to mark up some textual content, which may contain an arbitrary amount of further well-bracketed markup. Informally, the more tags we add, the more the text is structured, and in some sense its "formal organization" will also increase. Note that we have not explicitly dealt with XML/HTML attributes, as they can be seen as common tagged elements and thus modeled as semistructured expressions. On the other hand, without loss of generality, other XML/HTML features such as namespaces, DTDs and/or schemas, that are not relevant to this work are not conveyed by our notion of semistructured expression.

In the literature, slightly different formalisms have been introduced for modeling XML and HTML documents, e.g. in [1] semistructured expressions are directed graphs which can deal with crossing references. Nevertheless, we prefer the hierarchical representation which does not cause any serious restriction in many practical contexts while it greatly simplifies our methodology.

[2] Note that symbol ϵ in the syntax given for semistructured expressions denotes the empty string and must not be confused with the empty marking ε.

3.0.2 Term representation.

Semistructured expressions are provided with a tree-like structure, therefore they can be conveniently translated into terms by applying the following straightforward transformation.

Definition 3.2 Let $\mathtt{e}$ be a semistructured expression over $\mathcal{T}ext$ and $\mathcal{T}ag$. Then, $\mathtt{e}$ is represented by a term of the Herbrand universe $\tau(\mathcal{T}ext \cup \mathcal{T}ag)$ by the translation
$s\_to\_t\colon \mathcal{S}(\mathcal{T}ext, \mathcal{T}ag) \to \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ defined as follows:

$$s\_to\_t(\mathtt{e}) = \begin{cases} \mathtt{w} & \text{if } \mathtt{e} \equiv \mathtt{w} \in \mathcal{T}ext \\ \mathtt{t}(s\_to\_t(\mathtt{e}_1), \ldots, s\_to\_t(\mathtt{e}_n)) & \text{if } \mathtt{e} \equiv \texttt{<t>}\ \mathtt{e}_1 \ldots \mathtt{e}_n \texttt{</t>} \end{cases}$$

Example 3.3 Consider again semistructured expression of Example 3.1. Then, the term $\mathtt{p}$ computed by function s_to_t for that semistructured expression is

```
members(
   member(name(mario),surname(rossi),status(professor)),
   member(name(franca),surname(bianchi),status(technician)),
   member(name(giulio),surname(verdi),status(Student))
)
```

To summarize, a Web page, which is coded as an HTML/XML document, can be represented as a semistructured expression, which is then easily translated into a corresponding term of a suitable term algebra. Therefore, in the remaining of this work, a Web page is modeled by a term in $\tau(\mathcal{T}ext \cup \mathcal{T}ag)$. Besides, a *marked* Web page is defined as $\mu(\mathtt{p})$, where $\mathtt{p} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ and μ is a valid marking for $\mathtt{p}$. A *Web site* is a finite collection of marked Web pages $\{\varepsilon(\mathtt{p}_1) \ldots \varepsilon(\mathtt{p}_n)\}$. In the following, we will also consider terms of the non-ground term algebra $\tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V})$, which may contain variables. An element $\mathtt{s} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V})$ is called *Web page template*. $\mu(\mathtt{s})$ is a *marked* Web page template, when $\mathtt{s} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V})$ and μ is a valid marking for $\mathtt{s}$. In our methodology, (marked) Web page templates are used for specifying properties on Web sites as described in the following section.

4 Web specification language

In the following, we present a term rewriting specification language, which is helpful to express properties about the content and the structure of a given Web site. Roughly speaking, a specification is a finite set of rules, where the terms in the left-hand side and in the right-hand side of each rule represent (eventually marked) Web page templates. The operational mechanism, formalized in Section 5, is based on a novel rewriting-based mechanism, which is able to extract partial structure from a term, and then rewrite it.

Formally, Web site specifications are as follows.

Definition 4.1 A *rule* is a pair of terms $\mathtt{l} \rightharpoonup \mu(\mathtt{r})$ such that $\mathtt{l}, \mathtt{r} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V})$, $\mathtt{l}$ is linear, $Var(\mathtt{r}) \subseteq Var(\mathtt{l})$ and μ is a valid marking for $\mathtt{r}$. A *Web site specification* $\mathtt{I}$ is a finite set of rules $\{\mathtt{l}_1 \rightharpoonup \mu_1(\mathtt{r}_1), \ldots, \mathtt{l}_n \rightharpoonup \mu_n(\mathtt{r}_n)\}$.

Given a Web specification $\mathtt{I}$, we denote the set of all left-hand sides (right-hand sides disregarding markings) of rules in $\mathtt{I}$ by $\mathtt{Lhs}_\mathtt{I}$ ($\mathtt{Rhs}_\mathtt{I}$, respectively). In symbols, $\mathtt{Lhs}_\mathtt{I} = \{\mathtt{l} \mid \mathtt{l} \rightharpoonup \mu(\mathtt{r}) \in \mathtt{I}\}$ and $\mathtt{Rhs}_\mathtt{I} = \{\mathtt{r} \mid \mathtt{l} \rightharpoonup \mu(\mathtt{r}) \in \mathtt{I}\}$.

The following example illustrates the definition of a Web specification. Marks are introduced by the user to help locating errors. We do not take care of marks for the time being but postpone the formal handling of marking information and the description of the verification framework to Section 6.

Example 4.2 Consider the following Web specification, which models some required properties of a research group Web site containing information about group members affiliation, scientific publications and personal data.

$$\mathtt{member(name(X), surname(Y))} \rightharpoonup \underline{\mathtt{hpage}}\mathtt{(name(X), }\underline{\mathtt{surname}}\mathtt{(}\underline{\mathtt{Y}}\mathtt{), status)}$$
$$\mathtt{hpage(status(professor))} \rightharpoonup \underline{\mathtt{hpage}}\mathtt{(}\underline{\mathtt{status}}\mathtt{(}\underline{\mathtt{professor}}\mathtt{), teaching)}$$
$$\mathtt{pubs(pub(name(X), surname(Y)))} \rightharpoonup \underline{\mathtt{member}}\mathtt{(name(X), }\underline{\mathtt{surname}}\mathtt{(}\underline{\mathtt{Y}}\mathtt{))}$$

First rule formalizes the following property: if there is a Web page containing a member list, then for each member, a home page exists containing (at least) the name, the surname and the status of this member. Second rule states that whenever a home page of a professor is recognized, then that page must also include some teaching information. Finally, the third rule specifies that whenever there exists a Web page containing information about scientific publications, each author of a publication should be a member of the research group.

Informally, rules of a Web specification formalize conditions to be fulfilled by a given Web site. Intuitively, the interpretation of a rule $\mathtt{l} \rightharpoonup \mu(\mathtt{r})$ w.r.t. a Web site $\mathtt{W}$ is as follows: if (an instance of) $\mathtt{l}$ is recognized in $\mathtt{W}$, also (an instance of) $\mathtt{r}$ must be recognized in a subset of $\mathtt{W}$, which is determined by computing the sets of all Web pages which embed (an instance of) the marked part of $\mathtt{r}$. This mechanism is formalized by partial rewriting.

5 Partial Rewriting

In order to mechanize the intended semantics of Web specification rules, we first devise a mechanism which is able to recognize the structure and the

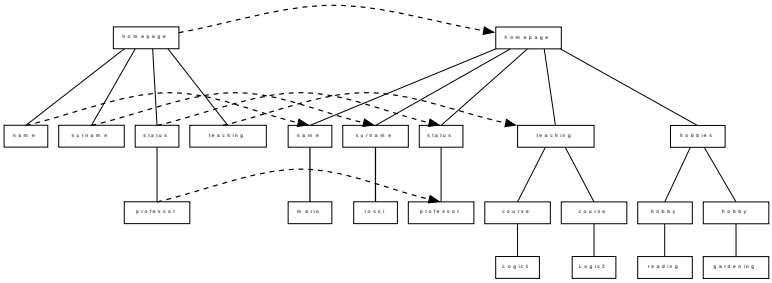

Fig. 1. Page simulation between $\mathtt{p}_1$ and $\mathtt{p}_2$.

labeling of a given Web page template inside a particular page of the Web site. This is provided by page simulation.

5.1 Page Simulations

The notion of *page simulation* for Web pages allows us to analyze and extract the partial structure of the Web site which is subject to verification.

Roughly speaking, a Web page $\mathtt{p}_1$ is simulated by a Web page $\mathtt{p}_2$, if the tree-structure of $\mathtt{p}_1$ is "embedded" into the tree-structure of $\mathtt{p}_2$. In other words, a simulation of a Web page (i.e. a labelled tree) $\mathtt{p}_1$ in a Web page $\mathtt{p}_2$ can be seen as a relation among the nodes of $\mathtt{p}_1$ and the nodes of $\mathtt{p}_2$ which preserves the edges and the labelings. Before formalizing the idea, we illustrate it by means of a rather intuitive example.

Example 5.1 Consider the following Web pages (called $\mathtt{p}_1$ and $\mathtt{p}_2$, respectively):

```
hpage(name,surname,status(professor),teaching)
```

```
hpage(name(mario),surname(rossi),status(professor),
      teaching(course(logic1),course(logic2)),
      hobbies(hobby(reading),hobby(gardening)))
```

Looking at Figure 1, we observe that the structure of $\mathtt{p}_1$ can be recognized inside the structure of $\mathtt{p}_2$ by considering the relation among nodes of $\mathtt{p}_1$ and nodes of $\mathtt{p}_2$ which is described by the dashed arrows in the figure. This relation essentially provides the so-called *simulation of* $\mathtt{p}_1$ *in* $\mathtt{p}_2$. Note that vice-versa does not hold: no relations can be found among nodes of $\mathtt{p}_2$ and nodes of $\mathtt{p}_1$, which "embed" the structure of $\mathtt{p}_2$ into $\mathtt{p}_1$. In other words, there does not exist a simulation of $\mathtt{p}_2$ in $\mathtt{p}_1$.

Simulations have been used in a number of works dealing with querying and transformation of semistructured data. For instance, [1,14] propose some techniques based on simulation for analyzing semistructured data w.r.t.

a given schema. The language Xcerpt [7,6] is a (logic) query language for XML and semistructured documents which implements a sort of unification by exploiting the notion of graph simulation. Other approaches involving simulation, or closely related notions, have been employed to measure similarity among semistructured documents [4]. To keep our framework simple, we do not consider a semantic change/load for labels; this would require to introduce ontologies, which are outside the scope of the paper.

Basically, the reason why simulations are successfully employed in the implementation of these kinds of manipulation and querying methods is twofold. Firstly, it is a simple and powerful technique to extract and recognize the partial structure of a document; secondly, there are several efficient algorithms to compute (graph and tree) simulations (see [15]).

In the following, we provide our notion of simulation which is a slight adaptation of the one given in [6] to consider Web page templates: we generalize the usual label relation to cope with the case when variables are used as labels, in the following definition.

Definition 5.2 Let $\mathtt{s}_1 \equiv (V_1, E_1, r_1, \mathit{label}_1)$, $\mathtt{s}_2 \equiv (V_2, E_2, r_2, \mathit{label}_2)$ be two Web page templates in $\tau(\mathcal{T}\mathit{ext} \cup \mathcal{T}\mathit{ag}, \mathcal{V})$. The *label* relation $\sim \subseteq V_1 \times V_2$ is defined as follows:

$$v_1 \sim v_2 \quad \text{iff} \quad \mathit{label}_1(v_1) = \mathit{label}_2(v_2) \text{ or } \mathit{label}_1(v_1) \in \mathcal{V}.$$

Definition 5.3 Let $\mathtt{s}_1 \equiv (r_1, V_1, E_1, r_1, \mathit{label}_1)$, $\mathtt{s}_2 \equiv (r_2, V_2, E_2, r_2, \mathit{label}_2)$ be two Web page templates in $\tau(\mathcal{T}\mathit{ext} \cup \mathcal{T}\mathit{ag}, \mathcal{V})$ and $\sim \subseteq V_1 \times V_2$ be the corresponding label relation. A *page simulation* of $\mathtt{s}_1$ in $\mathtt{s}_2$ w.r.t $\sim$ is a relation $\mathbf{S} \subseteq V_1 \times V_2$ such that, for each $v_1 \in V_1, v_2 \in V_2$

(i) $r_1 \mathbf{S} r_2$;

(ii) $v_1 \mathbf{S} v_2 \Rightarrow v_1 \sim v_2$;

(iii) $v_1 \mathbf{S} v_2 \wedge (v_1, v_1') \in E_1 \Rightarrow \exists\, v_2' \in V_2,\ v_1' \mathbf{S} v_2' \wedge (v_2, v_2') \in E_2$.

We define the *projection* of a simulation $\mathbf{S}$ of $\mathtt{s}_1$ in $\mathtt{s}_2$ w.r.t $\sim$ as $\pi(\mathbf{S}) = \{v_2 \mid (v_1, v_2) \in \mathbf{S}\}$.

Roughly speaking, Definition 5.3 ensures two degrees of similarity between Web page templates, not only w.r.t. the labelings but also w.r.t. the structures of the templates. On the one hand, Condition (2) of Definition 5.3 formalizes the similarity w.r.t labelings, that is, any pair of nodes (v, v') in a page simulation $\mathbf{S}$ of $\mathtt{s}_1$ in $\mathtt{s}_2$ have the same label, otherwise node v must be labelled by a variable, which somehow means that the label of v can be seen as a generalization of any concrete label of v'. Finally, Condition (1) and Condition (3) provide a relation between the tree structure of $\mathtt{s}_1$ and the tree

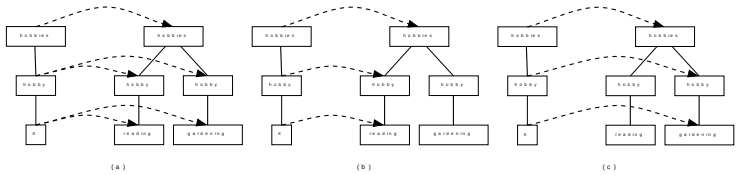

Fig. 2. non-minimal and minimal simulations

structure of $\mathtt{s}_2$.

Note that simulations are just relations among nodes of two given Web page templates. For our purposes, we are interested in simulations which are injective mappings from nodes of a given Web page template to nodes of another Web page template. As it will be apparent later, those simulations allow us to project the structure of a Web page template into another one, thus performing a sort of "partial" pattern matching between templates, which will be exploited to formulate our verification technique.

In the following, we define a subclass of simulations called minimal simulations.

Definition 5.4 Let $\mathtt{s}_1 \equiv (V_1, E_1, r_1, label_1)$, $\mathtt{s}_2 \equiv (V_2, E_2, r_2, label_2)$ be two Web page templates in $\tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V})$. A page simulation $\mathbf{S}$ of $\mathtt{s}_1$ in $\mathtt{s}_2$ w.r.t. $\sim$ is *minimal* if there are no page simulations $\mathbf{S}'$ of $\mathtt{s}_1$ in $\mathtt{s}_2$ w.r.t. $\sim$ such that $\mathbf{S}' \subseteq \mathbf{S}$.

Let us see an example which illustrates the notion of minimal simulation.

Example 5.5 Let us consider the following Web page templates $\mathtt{s}_1$ and $\mathtt{s}_2$: `hobbies(hobby(X))`, `hobbies(hobby(reading),hobby(gardening))`. In Figure 2(a), the dashed arrows represent a non-minimal simulation of $\mathtt{s}_1$ in $\mathtt{s}_2$, while in Figures 2(b) and 2(c) two minimal simulations of $\mathtt{s}_1$ in $\mathtt{s}_2$ are depicted. Note that the last two simulations are mappings.

Lemma 5.6 *Let* $\mathtt{s}_1 \equiv (V_1, E_1, r_1, label_1)$, $\mathtt{s}_2 \equiv (V_2, E_2, r_2, label_2)$ *be two Web page templates in* $\tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V})$. *A minimal page simulation* $\mathbf{S}$ *of* $\mathtt{s}_1$ *in* $\mathtt{s}_2$ *w.r.t.* $\sim$ *is a mapping* $\mathbf{S} : V_1 \to V_2$.

Minimal simulations do not guarantee that the tree structure of a given Web page template can be recognized inside another template. For this purpose, we need to furtherly restrict our class of simulations. Let us see an example.

Example 5.7 Consider Web page templates $\mathtt{s}_1 \equiv \mathtt{f(X,Y)}$ and $\mathtt{s}_2 \equiv \mathtt{f(a)}$. Note that there exists a minimal page simulation of $\mathtt{s}_1$ in $\mathtt{s}_2$ w.r.t. $\sim$ (see Figure 3), but the tree structure of $\mathtt{s}_1$ cannot be recognized as part of $\mathtt{s}_2$, e.g.

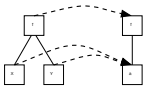

Fig. 3. minimal non-injective simulation

the vertex with label f in $\mathtt{s}_1$ has two outgoing edges, while the corresponding vertex in $\mathtt{s}_2$ has only one.

To solve the problem presented in Example 5.7, we simply restrict ourselves to consider minimal *injective* page simulations, which provide a one-to-one correspondence among edges of the two considered Web page templates.

It is not difficult to demonstrate that minimal injective simulations are particular instances of Kruskal's *embeddings* [5] w.r.t. the relation $\sim$. In other words, a minimal injective page simulation of $\mathtt{s}_1$ in $\mathtt{s}_2$ w.r.t. $\sim$ exists iff $\mathtt{s}_1$ is embedded into $\mathtt{s}_2$ w.r.t. $\sim$, i.e., we are able to find out the structure and the labeling of $\mathtt{s}_1$ inside $\mathtt{s}_2$. Note that the minimal simulation of $\mathtt{s}_1$ in $\mathtt{s}_2$ depicted in Figure 3 is not injective and thus no embedding of $\mathtt{s}_1$ into $\mathtt{s}_2$ exists. Instead, Figures 2(b) and 2(c) illustrate two minimal injective simulations, that is, two embeddings between Web page templates.

5.2 Rewriting Web page templates

Definition 5.8 Let $\mathtt{s}_1 \equiv (V_1, E_1, r_1, label_1)$, $\mathtt{s}_2 \equiv (V_2, E_2, r_2, label_2) \in \tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V})$. We say that $\mathtt{s}_2$ *partially matches* $\mathtt{s}_1$ via substitution σ iff

(i) there exists a minimal injective simulation $\mathbf{S}$ of $\mathtt{s}_1$ in $\mathtt{s}_2$ w.r.t. $\sim$;

(ii) for each $(v, v') \in \mathbf{S}$ such that $label(v) = X \in \mathcal{V}$, $\sigma(X) = (\mathtt{s}_{2|v'})$.

In Definition 5.8, we consider only minimal injective simulations between Web page templates $\mathtt{s}_1$ and $\mathtt{s}_2$, since this trivially ensures the existence of a substitution σ such that there exists a simulation of $\mathtt{s}_1\sigma$ in $\mathtt{s}_2$ w.r.t. $\sim$; in other words, $\mathtt{s}_1\sigma$ is embedded into $\mathtt{s}_2$.

Example 5.9 Consider again Example 5.5. We have that $\mathtt{s}_2$ partially matches $\mathtt{s}_1$ via $\{X/reading\}$ (see Figure 2(b)) and $\mathtt{s}_2$ partially matches $\mathtt{s}_1$ via $\{X/gardening\}$ (see Figure 2(c)). Note that performing partial matching by the non-minimal simulation of Figure 2(a) would produce $\sigma \equiv \{X/reading, X/gardening\}$, which is not a substitution.

Now we are ready to define a partial rewrite relation between marked Web page templates.

Definition 5.10 Let $\mathtt{s} \equiv (V, E, r, label)$, $\mathtt{t} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V})$. Let μ_1 and μ_2 be two valid markings for $\mathtt{s}$ and $\mathtt{t}$, respectively. Then, $\mu_1(\mathtt{s})$ *partially*

rewrites to $\mu_2(\mathtt{t})$ via rule $r \equiv \mathtt{l} \rightharpoonup \mu(\mathtt{r})$ and substitution σ (in symbols, $\mu_1(\mathtt{s}) \rightharpoonup_r^\sigma \mu_2(\mathtt{t})$) iff there exists $v \in V$ such that

(i) $\mathtt{s}_{|v}$ *partially matches* $\mathtt{l}$ via σ;

(ii) $\mathtt{t} = \mathtt{r}\sigma$.

(iii) Let $\mathtt{r} \equiv (V_\mathtt{r}, E_\mathtt{r}, r, label_\mathtt{r})$ and $\mathtt{r}\sigma \equiv (V_{\mathtt{r}\sigma}, E_{\mathtt{r}\sigma}, r, label_{\mathtt{r}\sigma})$. For each $v \in V_{\mathtt{r}\sigma}$,

$$\mu_2(v) = \begin{cases} \mu(v) & \text{if } \ v \in (V_\mathtt{r} \cap V_{\mathtt{r}\sigma}) \\ \mu(v') & \text{if } \ v \in (V_{\mathtt{r}\sigma} \setminus V_\mathtt{r}) \wedge (\exists\, v' \in V_\mathtt{r}, v \geq v', label_\mathtt{r}(v') \in Var(\mathtt{r})) \end{cases}$$

When rule r and substitution σ are understood, we simply write $\mu_1(\mathtt{s}) \rightharpoonup \mu_2(\mathtt{t})$.

It is worth noting that we provide a notion of partial rewriting in which the context of the selected reducible expression $\mathtt{s}_{|v}$ of the Web page template which is rewritten is disregarded after the rewrite step (see point (2) of Definition 5.10). Roughly speaking, given a Web specification rule $\mathtt{l} \rightharpoonup \mu(\mathtt{r})$, partial rewriting allows us to extract a subpart of a given Web page (template) $\mathtt{s}$, which partially matches $\mathtt{l}$, and to replace $\mathtt{s}$ by an instance of $\mathtt{r}$; namely, $\mathtt{r}\sigma$ (see points (1) and (2) of Definition 5.10). Point (3) of Definition 5.10 establishes that rewritten templates inherit markings from the right-hand sides of the applied rules. More precisely,

- each vertex of $\mathtt{r}\sigma$, which is not affected by substitution σ, maintains the same marking of $\mathtt{r}$;
- each vertex, which belongs to a subterm of $\mathtt{r}\sigma$ replacing a variable $\underline{X}$ of $\mathtt{r}$, is marked *yes*;
- each vertex, which belongs to a subterm of $\mathtt{r}\sigma$ replacing a variable X of $\mathtt{r}$, is marked *no*.

Example 5.11 Consider the Web page $\mathtt{p}$ of Example 3.3 and the first rule $\mathtt{r}_1$ of the Web specification of Example 4.2. Then, Web page template $\varepsilon(\mathtt{p})$ partially rewrites to the following three Web pages by applying $\mathtt{r}_1$.

$$\varepsilon(\mathtt{p}) \rightharpoonup_{\mathtt{r}_1} \underline{\mathtt{hpage}}(\mathtt{name}(\mathtt{mario}), \underline{\mathtt{surname}}(\underline{\mathtt{rossi}}), \mathtt{status})$$
$$\varepsilon(\mathtt{p}) \rightharpoonup_{\mathtt{r}_1} \underline{\mathtt{hpage}}(\mathtt{name}(\mathtt{franca}), \underline{\mathtt{surname}}(\underline{\mathtt{bianchi}}), \mathtt{status})$$
$$\varepsilon(\mathtt{p}) \rightharpoonup_{\mathtt{r}_1} \underline{\mathtt{hpage}}(\mathtt{name}(\mathtt{giulio}), \underline{\mathtt{surname}}(\underline{\mathtt{verdi}}), \mathtt{status})$$

Roughly speaking, markings in the right-hand sides of the rules allow us to find sets of Web pages, which might be incomplete or missing. Then, real buggy pages are detected inside these sets. We formalize the idea in the following section.

6 The verification framework

In the following, we show how simulation and partial rewriting can be applied to verify a given Web site $\mathtt{W}$ w.r.t. a Web specification $\mathtt{I}$. Essentially, the main idea is to compute the set of all possible marked Web pages that can be derived from $\mathtt{W}$ via $\mathtt{I}$ by means of partial rewriting. These marked Web pages can be thought of as requirements to be fulfilled by $\mathtt{W}$. Then, we check whether the computed requirements are satisfied by $\mathtt{W}$ by using simulation and marking information. In summary, the method works in two steps, which are repeatedly applied as described in the following.

(i) Compute the set of requirements $\mathtt{Req}_{\mathtt{I},\mathtt{W}}$ for $\mathtt{W}$ w.r.t. $\mathtt{I}$

(ii) Check $\mathtt{Req}_{\mathtt{I},\mathtt{W}}$ in $\mathtt{W}$.

6.1 Computing the set of requirements

Let us introduce the following operator.

Definition 6.1 Let $\mathtt{T}$ be a set of marked Web page templates and $\mathtt{I}$ be a Web specification. Then,

$$\mathtt{R}_\mathtt{I}(\mathtt{T}) = \mathtt{T} \cup \{\mu_2(\mathtt{s}_2) \mid \exists\, \mu_1(\mathtt{s}_1) \in \mathtt{T}, r \equiv \mathtt{l} \rightharpoonup \mu(\mathtt{r}) \in \mathtt{I} \text{ s.t. } \mu_1(\mathtt{s}_1) \rightharpoonup_r \mu_2(\mathtt{s}_2)\}$$

Roughly speaking, the operator in Definition 6.1 computes all marked templates which result from partial rewriting the Web page templates of $\mathtt{T}$ by using the Web specification $\mathtt{I}$, and returns the union of the resulting set and $\mathtt{T}$. By repeatedly applying this operator, it is possible to compute all marked Web pages that can be derived from an initial Web site after an arbitrary number of partially rewriting steps. For this purpose, we formalize the *ordinal powers* of the operator $\mathtt{R}_\mathtt{I}$ w.r.t. a Web site $\mathtt{W}$ as follows: $\mathtt{R}_\mathtt{I} \uparrow^\mathtt{W} 0 = \mathtt{W}$, $\mathtt{R}_\mathtt{I} \uparrow^\mathtt{W} n = \mathtt{R}_\mathtt{I}(\mathtt{R}_\mathtt{I} \uparrow^\mathtt{W} (n-1))$, $n > 0$.

It is immediate to demonstrate that the operator $\mathtt{R}_\mathtt{I}$ is continuous on the lattice consisting of the powerset of the term algebra of the marked Web page templates ordered by set inclusion. This ensures that a least fixpoint of $\mathtt{R}_\mathtt{I}$ exists and can be reached after ω applications of $\mathtt{R}_\mathtt{I}$, that is, $\mathtt{R}_\mathtt{I} \uparrow^\mathtt{W} \omega$ where ω is the first infinite ordinal. Moreover, the least fixpoint of $\mathtt{R}_\mathtt{I}$ contains all the marked Web pages derivable from Web pages in $\mathtt{W}$ via $\mathtt{I}$.

Now, recalling the interpretation of the rules of the Web site specification given in Section 4, Web pages derived by the application of a Web specification must be recognized as (part of) some Web page in the Web site. Therefore, those Web pages in the least fixpoint of $\mathtt{R}_\mathtt{I}$ which are not in $\mathtt{W}$ can be intended as requirements to be fulfilled by $\mathtt{W}$. Thus, we define the *set of requirements*

for $\mathtt{W}$ w.r.t. $\mathtt{I}$ as $\mathtt{Req}_{\mathtt{I},\mathtt{W}} = \mathit{lfp}(\mathtt{R}_\mathtt{I}) \setminus \mathtt{W}$, where $\mathit{lfp}(\mathtt{R}_\mathtt{I})$ is the least fixpoint of the operator $\mathtt{R}_\mathtt{I}$.

Clearly, the fixpoint of $\mathtt{R}_\mathtt{I}$ (and hence $\mathtt{Req}_{\mathtt{I},\mathtt{W}}$) for an arbitrary Web specification might be infinite. Consider for instance the following example.

Example 6.2 Let $\mathtt{W} \equiv \{\mathtt{h}(\mathtt{g}(0), \mathtt{f}(0))\}$ be a Web site and $\mathtt{I} \equiv \{\mathtt{h}(\mathtt{g}(\mathtt{X})) \rightharpoonup \mathtt{h}(\mathtt{g}(\mathtt{g}(\mathtt{X})))\}$ be a Web specification. Then,

$$\mathtt{Req}_{\mathtt{I},\mathtt{W}} = \{\mathtt{h}(\mathtt{g}(\mathtt{g}(0))), \mathtt{h}(\mathtt{g}(\mathtt{g}(\mathtt{g}(0)))), \mathtt{h}(\mathtt{g}(\mathtt{g}(\mathtt{g}(\mathtt{g}(0))))), \ldots\}$$

is an infinite set of requirements which is infinite.

Fortunately, the computation of the set of requirements is finite for some interesting classes of Web specifications. Trivially, non-recursive specifications allow to reach $\mathit{lfp}(\mathtt{R}_\mathtt{I})$ after a finite number of applications of $\mathtt{R}_\mathtt{I}$, i.e., $\mathit{lfp}(\mathtt{R}_\mathtt{I}) = \mathtt{R}_\mathtt{I} \uparrow^{\mathtt{W}} k$, $k \in \mathbb{N}$. However, non-recursive definitions are not expressive enough for verification purposes, since some relevant conditions about Web sites cannot be formalized without resorting to recursion; e.g., some properties stated in Example 4.2 cannot be formulated by using a non-recursive specification.

In the following, we define a class of recursive Web specifications for which the set of requirements is finite. Basically, the idea is to consider those specifications for which the computation of the least fixpoint only generates Web pages whose size is bounded.

The following definition formalizes the considered class of Web site specifications.

Definition 6.3 A Web specification $\mathtt{I}$ is *bounded* iff, for each $\mathtt{l} \equiv (V_1, E_1, r_1, \mathit{label}_1) \in \mathtt{Lhs}_\mathtt{I}$, $\mathtt{r} \equiv (V_2, E_2, r_2, \mathit{label}_2) \in \mathtt{Rhs}_\mathtt{I}$ and each minimal injective simulation $\mathbf{S}$ of $\mathtt{l}$ in $\mathtt{r}_{|v}$ w.r.t. $\sim$, $v \in V_2$, the following property holds

$$\text{if } v_2 \in \pi(\mathbf{S}) \text{ and } \mathit{label}_2(v_2) \in \mathit{Var}(\mathtt{r}_{|v}), \text{ then for all } v_1 \in V_1 \text{ s.t.}$$
$$\mathit{label}_1(v_1) \in \mathit{Var}(\mathtt{l}),\ \mathit{depth}(\mathtt{r}_{|v}, v_2) = \mathit{depth}(\mathtt{l}, v_1).$$

Roughly speaking, Definition 6.3 states that, whenever a left-hand side $\mathtt{l}$ of a rule is simulated by (a subterm of) the right-hand side $\mathtt{r}$ of a (possibly different) rule, then no variables in the substructure of $\mathtt{r}$ which is recognized by simulation must be located at positions which are deeper than all the positions of the variables in $\mathtt{l}$.

Example 6.4 Consider again the specification $\mathtt{I}$ in Example 6.2. The left-hand side of the rule $\mathtt{h}(\mathtt{g}(\mathtt{X})) \rightharpoonup \mathtt{h}(\mathtt{g}(\mathtt{g}(\mathtt{X})))$ is simulated by its own right-hand

side. Moreover, variable X in the right-hand side is located at depth 3, while the unique variable in the left-hand side is at depth 2. Thus, I is not bounded.

Now, take into account specification

$$\mathtt{I}' \equiv \{\mathtt{m(n(X))} \rightharpoonup \mathtt{h(n(X),s(s(X)))}, \mathtt{h(n(X))} \rightharpoonup \mathtt{m(n(X),t)}\}.$$

Then, m(n(X)) is simulated by m(n(X),t) and h(n(X)) is simulated by h(n(X),s(s(X))). In both cases, variables occurring in the substructures of the right-hand sides which are recognized by simulation and variables of the respective left-hand sides are located at the same depth. Therefore, the Web specification I′ is bounded.

For bounded Web specifications, the least fixpoint of the operator $\mathtt{R}_\mathtt{I}$ is finite as stated by the next proposition. This provides an effective method for computing the set of requirements $\mathtt{Req}_{\mathtt{I},\mathtt{W}}$.

Proposition 6.5 *Let* I *be a bounded Web specification and* W *be a Web site. Then, there exists* $k \in \mathbb{N}$ *such that* $lfp(\mathtt{R}_\mathtt{I}) = \mathtt{R}_\mathtt{I} \uparrow^{\mathtt{W}} k$.

Example 6.6 Consider the bounded Web specification I of Example 4.2 and the following Web site W:

```
W = {members(member(name(mario),surname(rossi),status(professor)),
             member(name(franca),surname(bianchi),status(technician)),
             member(name(anna),surname(gialli),status(professor)),
             member(name(giulio),surname(verdi),status(student))),
     hpage(name(mario),surname(rossi),phone(3333),status(professor),
           hobbies(hobby(reading),hobby(gardening))),
     hpage(name(franca),surname(bianchi),status(technician),phone(5555)),
     hpage(name(anna),surname(gialli),status(professor),phone(4444),
           teaching(course(algebra))),
     pubs(pub(name(mario),surname(rossi),title(blahblah1),year(2003)),
          pub(name(anna),surname(gialli),title(blahblah2),year(2002)))}
```

Then, the set of computed requirements $\mathtt{Req}_{\mathtt{I},\mathtt{W}}$ is

$$\begin{array}{l}\{\ \underline{\mathtt{hpage}}(\mathtt{name}(\mathtt{mario}), \underline{\mathtt{surname}}(\underline{\mathtt{rossi}}), \mathtt{status}),\\ \quad \underline{\mathtt{hpage}}(\mathtt{name}(\mathtt{franca}), \underline{\mathtt{surname}}(\underline{\mathtt{bianchi}}), \mathtt{status}),\\ \quad \underline{\mathtt{hpage}}(\mathtt{name}(\mathtt{anna}), \underline{\mathtt{surname}}(\underline{\mathtt{gialli}}), \mathtt{status}),\\ \quad \underline{\mathtt{hpage}}(\mathtt{name}(\mathtt{giulio}), \underline{\mathtt{surname}}(\underline{\mathtt{verdi}}), \mathtt{status}),\\ \quad \underline{\mathtt{hpage}}(\underline{\mathtt{status}}(\underline{\mathtt{professor}}), \mathtt{teaching}),\\ \quad \underline{\mathtt{member}}(\mathtt{name}(\mathtt{mario}), \underline{\mathtt{surname}}(\underline{\mathtt{rossi}})),\\ \quad \underline{\mathtt{member}}(\mathtt{name}(\mathtt{anna}), \underline{\mathtt{surname}}(\underline{\mathtt{gialli}}))\ \}\end{array}$$

6.2 Checking requirements in Web sites

As we have seen in Section 5.1, simulation allows us to identify the structure of a given Web page (eventually, a template) into another. By taking advantage of this fact, we can develop a methodology, which is able to discover incompleteness errors in a given Web site w.r.t. a Web specification. Basically, the

idea is to verify the consistency of the Web site w.r.t. the set of requirements. To accomplish this task, we first use simulation for checking whether requirements are embedded into some Web page of the considered Web site and then exploit marking information in order to diagnose incompleteness errors in the Web site.

More precisely, our analysis allows us to discover two kinds of incompleteness errors: (1) Web pages which are missing in a Web site w.r.t. a given Web specification, (2) Web pages which are incomplete w.r.t a given Web specification.

Let us first consider the former class of errors.

Definition 6.7 Let $\mathtt{W}$ be a Web site, $\mathtt{I}$ be a bounded Web specification and $\mathtt{Req}_{\mathtt{I},\mathtt{W}}$ be the set of requirements for $\mathtt{W}$ w.r.t. $\mathtt{I}$. Let $\mu(\mathtt{e}) \in \mathtt{Req}_{\mathtt{I},\mathtt{W}}$. The *likely missed information set* w.r.t. $\mu(\mathtt{e})$ is defined as

$$LMIS_{\mu(\mathtt{e})} = \{\mathtt{p} \equiv (V, E, r, label) \in \mathtt{W} \mid \text{there is a minimal injective simulation of } mark(\mathtt{e}, \mu) \text{ in } \mathtt{p}_{|v} \text{ w.r.t. } \sim, \text{ with } v \in V\}.$$

Roughly speaking, this definition allows us to compute a subset of the Web site containing all the web pages which are simulated by the marked part of a given requirement. These web pages could be potentially incomplete w.r.t. the web specification, since they might not satisfy the considered requirement. Let us see an example.

Example 6.8 Let us consider the rule $\mathtt{r}$

$$\mathtt{hpage}(\mathtt{status}(\mathtt{professor})) \rightharpoonup \underline{\mathtt{hpage}}(\underline{\mathtt{status}}(\underline{\mathtt{professor}}), \mathtt{teaching})$$

and the website $\mathtt{W}$ of Example 6.6. Rule $\mathtt{r}$ allows us to check whether web pages of professors contain some teaching information. Clearly, requirements computed by this rule should be only checked in such web pages. For this purpose, we use the marking information in the rhs of $\mathtt{r}$ in order to focus on the professor web pages. Let us consider the requirement

$$\mu_1(\mathtt{e}_1) \equiv \underline{\mathtt{hpage}}(\underline{\mathtt{status}}(\underline{\mathtt{professor}}), \mathtt{teaching})$$

which can be derived from $\mathtt{W}$ by means of r. By applying Definition 6.7, we get

$$\begin{aligned} LMIS_{\mu_1(\mathtt{e}_1)} = \{(1)\ & \mathtt{hpage}(\ \mathtt{name}(\mathtt{mario}), \mathtt{surname}(\mathtt{rossi}),\\ & \mathtt{phone}(3333), \mathtt{status}(\mathtt{professor}),\\ & \mathtt{hobbies}(\mathtt{hobby}(\mathtt{reading}), \mathtt{hobby}(\mathtt{gardening}))),\\ (2)\ & \mathtt{hpage}(\ \mathtt{name}(\mathtt{anna}), \mathtt{surname}(\mathtt{gialli}),\\ & \mathtt{status}(\mathtt{professor}), \mathtt{phone}(4444),\\ & \mathtt{teaching}(\mathtt{course}(\mathtt{algebra}))) \}. \end{aligned}$$

which contains only professor web pages to be checked for incompleteness errors.

From Definition 6.7, we can easily derive that, whenever the likely missing information set is empty for a given requirement $\mu(\mathtt{e})$, $\mu(\mathtt{e})$ is not recognized in any Web page of the Web site. In other words,that requirement identifies a missing element in the Web site.

Definition 6.9 Let $\mathtt{W}$ be a Web site, $\mathtt{I}$ be a bounded Web specification and $\mathtt{Req}_{\mathtt{I},\mathtt{W}}$ be the set of requirements for $\mathtt{W}$ w.r.t. $\mathtt{I}$. Let $\mu(\mathtt{e}) \in \mathtt{Req}_{\mathtt{I},\mathtt{W}}$. Then, $\mu(\mathtt{e})$ is missing in $\mathtt{W}$ w.r.t. $\mathtt{I}$ iff $LMIS_{\mu(\mathtt{e})} = \emptyset$.

Let us see an example for clarifying our definitions.

Example 6.10 Consider again the set of requirements $\mathtt{Req}_{\mathtt{I},\mathtt{W}}$ computed in Example 6.6. Then, $\mu(\mathtt{e}) \equiv (\underline{\mathtt{hpage}}(\mathtt{name}(\mathtt{giulio}), \underline{\mathtt{surname}}(\underline{\mathtt{verdi}}), \mathtt{status}))$ is missing in $\mathtt{W}$ w.r.t. $\mathtt{I}$, since $LMIS_{\mu(\mathtt{e})} = \emptyset$. Indeed, the requirement $\mu(\mathtt{e})$ identifies a "group member" home page which does not appear in the Web site $\mathtt{W}$.

Let us consider now incompleteness errors which refer to incomplete pages, that is, Web pages in which some piece of information is lacking (e.g. missing items).

Definition 6.11 Let $\mathtt{W}$ be a Web site, $\mathtt{I}$ be a bounded Web specification and $\mathtt{Req}_{\mathtt{I},\mathtt{W}}$ be the set of requirements for $\mathtt{W}$ w.r.t. $\mathtt{I}$. Let $\mu(\mathtt{e}) \in \mathtt{Req}_{\mathtt{I},\mathtt{W}}$ and $\mathtt{p} \in \mathtt{W}$. Then, $\mathtt{p} \equiv (V, E, r, \mathit{label})$ is incomplete w.r.t. $\mu(\mathtt{e})$ iff

- $\mathtt{p} \in LMIS_{\mu(\mathtt{e})}$;
- there is a minimal injective simulation of $\mathit{mark}(\mathtt{e}, \mu)$ in $\mathtt{p}_{|v}$ w.r.t. $\sim$, with $v \in V$, s. t. there is no minimal injective simulation of $\mathtt{e}$ in $\mathtt{p}_{|v}$ w.r.t. $\sim$.

In this case, we will call $\mu(\mathtt{e})$ *incompleteness symptom* for $\mathtt{p}$.

Example 6.12 Recall the set of requirements $\mathtt{Req}_{\mathtt{I},\mathtt{W}}$ computed in Example 6.6. Then, consider the requirement

$$\mu_1(\mathtt{e}_1) \equiv (\underline{\mathtt{hpage}}(\underline{\mathtt{status}}(\underline{\mathtt{professor}}), \mathtt{teaching})),$$

we have that

$$\begin{aligned} LMIS_{\mu_1(\mathtt{e}_1)} = \{(1)\ & \mathtt{hpage}(\ \mathtt{name}(\mathtt{mario}), \mathtt{surname}(\mathtt{rossi}),\\ & \mathtt{phone}(3333), \mathtt{status}(\mathtt{professor}),\\ & \mathtt{hobbies}(\mathtt{hobby}(\mathtt{reading}), \mathtt{hobby}(\mathtt{gardening}))),\\ (2)\ & \mathtt{hpage}(\ \mathtt{name}(\mathtt{anna}), \mathtt{surname}(\mathtt{gialli}),\\ & \mathtt{status}(\mathtt{professor}), \mathtt{phone}(4444),\\ & \mathtt{teaching}(\mathtt{course}(\mathtt{algebra}))\ )\ \}. \end{aligned}$$

Now, by applying Definition 6.11, we detect that Web page (1) is incomplete w.r.t. $\mu_1(\mathtt{e}_1)$, which is therefore an incompleteness symptom for (1). In fact, Web page (1) lacks teaching information.

Finally, the remaining requirements do not give rise to further errors.

It is worth pointing out that our verification framework is able to detect both the erroneous Web pages and the cause of the detected errors (i.e., the so-called incompleteness symptoms). This allows us not only to locate bugs and inconsistencies w.r.t. a given specification, but also to easily repair them by comparing incomplete pages to incompleteness symptoms, since the latter provides the missing information which is needed to complete the erroneous Web pages.

7 Implementation

The basic methodology presented so far has been implemented in the preliminary prototype system VERDI (VErification and Rewriting for Debugging Internet sites), which is written in DrScheme v205 [18] and is publicly available together with a set of tests at `http://www.dimi.uniud.it/~demis/#software`.

The implementation consists of about 80 function definitions (approximately 1000 lines of source code). VERDI includes a parser for semistructured expressions and Web specifications, and several modules implementing the user interface, the partial rewriting mechanism and the verification technique. The system allows the user to load a Web site consisting of a finite set of semistructured expressions together with a Web specification. Additionally, he/she can inspect the loaded data and finally check the Web pages w.r.t the Web site specification. The user interface is guided by textual menus, which are (hopefully) self-explaining.

We tested the system on several Web site examples which can be found at the URL address mentioned above. In each considered test case, we were able to detect the errors (i.e. missing and incomplete Web pages) efficiently. For instance, it took less than one second the verification of the Web site of Example 6.6 w.r.t the Web specification of the Example 4.2, producing error messages when necessary.

8 Conclusions

Conceiving and maintaining Web sites is a difficult task. In this paper, we provide a rewriting-based, formal specification language which can be used to impose properties both on the structure (syntactic properties) and on the contents (semantic properties) of Web sites. The computation mechanism underlying this language is based on a novel rewriting-like technique, called *partial rewriting*, in which the traditional pattern matching mechanism is replaced with tree *simulation* [15]. In our methodology, Web sites are automatically checked w.r.t. a given Web specification in order to detect incomplete and

missing Web pages. Moreover, by analyzing the requirements not fulfilled by the Web site, we are also able to find out the missing information needed to repair the Web site. Our methodology exploits some marking information on terms which represent the Web pages to better locate the errors, which is provided by the user in advance. We have also discussed some implementation details of the preliminary system VERDI, a prototype implementation of the verification framework that we propose. Thus, we use the rewriting-based machinery as a common formalism for the specifications, for specifying our verification technique and for implementing the verification tool.

Finally, let us conclude by mentioning some directions for future work. We are currently extending our framework in order to provide a method for synthesizing the marking information semi-automatically. We also plan to extend the specification language in order to support the detection of regular expressions. This is useful to guarantee that proprietary or "forbidden" data are not displayed on the external version of the site (e.g. a number of credit card). On the practical side, we plan to supply a fully user-friendly system which can help Web administrators to design, check and maintain their Web sites. The system will be developed in Maude [9], which provides an efficient implementation of AC-rewriting, which can greatly simplify as well as improve the implementation of the partial-rewriting mechanism.

References

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web. From Relations to Semistructured Data and XML*. Morgan Kaufmann, 2000.

[2] M. Alpuente, D. Ballis, and M. Falaschi. A Rewriting-based Framework for Web Sites Verification. Technical Report 4, DSIC, UPV, 2004. Available at URL: http://www.dsic.upv.es/users/elp/papers.html.

[3] I. D. Baxter, F. Ricca, and P. Tonella. Web Application Transformations based on Rewrite Rules. *Information and Software Technology*, 44(13), 2002.

[4] E. Bertino, M. Mesiti, and G. Guerrin. A Matching Algorithm for Measuring the Structural Similarity between an XML Document and a DTD and its Applications. *Information Systems*, 29(1):23–46, 2004.

[5] M. Bezem. *TeReSe, Term Rewriting Systems*, chapter Mathematical background (Appendix A). Cambridge University Press, 2003.

[6] F. Bry and S. Schaffert. Towards a Declarative Query and Transformation Language for XML and Semistructured Data: Simulation Unification. In *Proc. of the Int'l Conference on Logic Programming (ICLP'02)*, volume 2401 of *LNCS*. Springer-Verlag, 2002.

[7] F. Bry and S. Sebastian Schaffert. The XML Query Language Xcerpt: Design Principles, Examples, and Semantics. Technical report, 2002. Available at: http://www.xcerpt.org.

[8] P. Buneman, S. B. Davidson, G. G. Hillebrand, and D. Suciu. A Query Language and Optimization Techniques for Unstructured Data. In *Proc. of ACM SIGMOD Int'l Conference on Management of Data (ICMD'96)*, 1996.

[9] M. Clavel, S. Eker, P. Lincoln, and J. Meseguer. Principles of Maude. In *Proc. of 1st Int'l Workshop on Rewriting Logic and its Applications (RWLW'96)*, volume 15 of ENTCS, pages 65–89. Elsevier, 1996, http://www.elsevier.com/locate/entcs/volume15.html

[10] A. Cortesi, A. Dovier, E. Quintarelli, and L. Tanca. Operational and abstract semantics of a graphical query language. *Theoretical Computer Science*, 275:521–560, 2002.

[11] N. Dershowitz and D. Plaisted. Rewriting. *Handbook of Automated Reasoning*, 1:535–610, 2001.

[12] T. Despeyroux and B. Trousse. Semantic Verification of Web Sites Using Natural Semantics. In *Proc. of 6th Conference on Content-Based Multimedia Information Access (RIAO'00)*, 2000.

[13] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. Verifying Integrity Constraints on Web Site. In *Proc. of Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99)*, volume 2, pages 614–619. Morgan Kaufmann, 1999.

[14] M. F. Fernandez and D. Suciu. Optimizing regular path expressions using graph schemas. In *Proc. of Int'l Conference on Data Engineering (ICDE'98)*, pages 14–23, 1998.

[15] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *IEEE Symposium on Foundations of Computer Science*, pages 453–462, 1995.

[16] H. Hosoya and B. C. Pierce. Regular Expression Pattern Matching for XML. In *Proc. of 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPLi01)*, volume 36(3) of *ACM SIGPLAN Notices*. ACM Press, 2001.

[17] C. Kirchner, Z. Qian, P. K. Singh, and J. Stuber. Xemantics: a Rewriting Calculus-Based Semantics of XSLT. Rapport de recherche A01-R-386, LORIA, 2001.

[18] PLT. DrScheme web site. Available at: http://www.drscheme.org.

[19] World Wide Web Consortium (W3C). HyperText Markup Language (HTML) 4.01, 1997. Available at: http://www.w3.org.

[20] World Wide Web Consortium (W3C). Extensible Markup Language (XML) 1.0, second edition, 1999. Available at: http://www.w3.org.