GVERDI-R A Tool for Repairing Faulty Web Sites

María Alpuente 1 Daniel Romero 1

DSIC, Universidad Politécnica de Valencia - Valencia, Spain

Demis Ballis 2

Dip. Matematica e Informatica - Udine, Italy

Moreno Falaschi 3

Dip. di Scienze Matematiche e Informatiche - Siena, Italy

Abstract

In this paper, we present a novel methodology for semi-automatically repairing faulty Web sites. We formulate a stepwise transformation procedure that achieves correctness and completeness of the Web site w.r.t. its formal specification while respecting the structure of the document (e.g. the schema of an XML document).

Key words: Formal Web site verification, specification languages, rewriting, simulation, repairing Web

1 Introduction

The increasing complexity of Web sites has turned their design and construction into a challenging problem. Systematic, formal approaches can bring many benefits to Web site construction, giving support for automated Web site verification and repairing. In our previous work on GVerdi [2,1], we presented a rewriting-like approach to Web site specification and verification. Our methodology allows us to specify the integrity conditions for the Web sites and then diagnose errors by computing the requirements not fulfilled

* This work has been partially supported by the EU (FEDER) and the Spanish MEC, under grant TIN 2004-7943-C04-02, by ICT for EU-India C.C.D. Project under grant ALA/95/23/2003/077-054, and by Generalitat Valenciana under grant GR03/025.
1 Email: {alpuente,dromero}@dsic.upv.es, Camino de Vera s/n, 46071 Valencia,Spain.
2 Email: demis@dimi.uniud.it, Via delle Scienze 206, 33100 Udine, Italy.
3 Email: moreno.falaschi@unisi.it, Pian dei Mantellini 44, 53100 Siena, Italy.

This paper was generated using LATEX macros provided by Electronic Notes in Theoretical Computer Science
by a given Web site, that is, by finding out incorrect/forbidden patterns and missing/incomplete Web pages.

In this paper, we aim to complement our methodology with a tool-independent technique for semi-automatically repairing the errors found during that verification phase. First, we formalize the kinds of errors that can be found in a Web site w.r.t. a Web Specification. Then, we classify the repair actions that can be performed to repair each kind of error.

The rest of the paper is structured as follows. Section 2 summarizes some preliminary definitions and notations about term rewriting systems. In Section 3, first we recall the Web verification framework of [2], which is based on tree simulation, and we categorize the different kinds of errors that can be found as an outcome of the verification technique. Section 4 describes our repairing methodology for faulty Web sites, while Section 5 concludes and discusses future work.

2 Preliminaries

We call a finite set of symbols alphabet. 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 $\Sigma$ denotes a set of function symbols, or signature. We consider varyadic signatures as in [6] (i.e., signatures in which symbols have an unbounded arity, that is, they may be followed by an arbitrary number of arguments). $\tau(\Sigma, \mathcal{V})$ and $\tau(\Sigma)$ denote the non-ground term algebra and the term algebra built on $\Sigma \cup \mathcal{V}$ and $\Sigma$. Terms are viewed as labelled trees in the usual way. Positions are represented by sequences of natural numbers denoting an access path in a term. By notation $w_1.w_2$, we denote the concatenation of position $w_1$ and position $w_2$. Positions are ordered by the prefix ordering. Given $S \subseteq \Sigma \cup \mathcal{V}$, $O_S(t)$ denotes the set of positions of a term $t$ which are rooted by symbols in $S$. 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 $\tau(\Sigma, \mathcal{V})$ satisfying the following conditions: $(i)$ $X_i \neq X_j$, whenever $i \neq j$, $(ii)$ $X_i \sigma = t_i$, $i = 1, \ldots, n$, and $(iii)$ $X \sigma = X$, for any $X \in \mathcal{V} \setminus \{X_1, \ldots, X_n\}$. By $\text{Var}(s)$ we denote the set of variables occurring in the syntactic object $s$. Term rewriting systems provide an adequate computational model for functional languages. In the sequel, we follow the standard framework of term rewriting (see [4,8]). A term rewriting system (TRS for short) is a pair $(\Sigma, R)$, where $\Sigma$ is a signature and $R$ is a finite set of reduction (or rewrite) rules.

3 Rewriting-based Web Verification

In this section, we briefly recall the formal verification methodology proposed in [2], which is able to detect forbidden/erroneous as well as missing information in a Web site. By executing a Web specification on a given Web site, we
Fig. 1. An example of a Web site for a research group

3.1 Denotation of Web sites

In our framework, a Web page is either an XML [11] or an XHTML [12] document, which we assume to be well-formed, since there are plenty of programs and online services which are able to validate XHTML/XML syntax and perform link checking (e.g. [10], [7]). XHTML/XML documents can be encoded as Herbrand terms. Web pages are provided with a tree-like structure, they can be straightforwardly translated into ordinary terms of a given term algebra $\tau(\text{Text} \cup \text{Tag})$. A Web site is a finite collection of ground terms \( \{p_1 \ldots p_n\} \).

3.2 Web specification language

A Web specification is a triple \((I_N, I_M, R)\), where \(I_N\) and \(I_M\) are finite sets of correctness and completeness rules, and the set \(R\) contains the rules for the definition of some auxiliary functions.

The set \(I_N\) describes constraints for detecting erroneous Web pages. A correctness rule has the following form: \(1 \rightarrow \text{error} | C\), with \(\text{Var}(C) \subseteq \text{Var}(1)\), where \(1\) is a term, \(\text{error}\) is a reserved constant, and \(C\) is a (possibly empty) finite sequence containing membership tests (e.g. \(X \in \text{rexp}\)) w.r.t. a given regular language\(^4\), and/or equations over terms.

The set of rules \(I_M\) specifies some properties for detecting incomplete/missing Web pages. A completeness rule is defined as \(1 \rightarrow r \langle q \rangle\), where \(1\) and \(r\) are terms and \(q \in \{E, A\}\). Completeness rules of a Web specification formalize the requirement that some information must be included in all or some pages of the Web site. We use attributes \(\langle A \rangle\) and \(\langle E \rangle\) to distinguish “universal” from “existential” rules. Right-hand sides of completeness rules can contain functions, which are defined in \(R\). Intuitively, the interpretation of a universal rule \(1 \rightarrow r \langle A \rangle\) (respectively, an existential rule \(1 \rightarrow r \langle E \rangle\)) w.r.t. a Web site \(W\) is as follows: if (an instance of) \(1\) is recognized in \(W\), also (an instance of)

\(^4\) Regular languages are represented by means of the usual Unix-like regular expressions syntax.
the irreducible form of $r$ must be recognized in all (respectively, some) of the Web pages which embed (an instance of) $r$.

**Example 3.1** Consider the Web specification which consists of the following completeness and correctness rules along with a term rewriting system defining the string concatenation function $++$, the arithmetic operators $+$ and $\ast$ on natural numbers and the relational operator $\leq$.

\[
\begin{align*}
\text{member(name}(X), \text{surname}(Y)) & \rightarrow \text{hpage(fullname}(X++Y), \text{status}(E) \\
\text{pubs}(\text{pub}(name}(X), \text{surname}(Y))) & \rightarrow \text{members(member(name}(X), \text{surname}(Y))) \langle E \rangle \\
\text{blink}(X) & \rightarrow \text{error} \\
\text{project}(\text{grant1}(X), \text{grant2}(Y), \text{total}(Z)) & \rightarrow \text{error} \mid X+Y \neq Z
\end{align*}
\]

This Web specification models some required properties for the Web site of Figure 1. First rule formalizes the following property: if there is a Web page containing a member list, then for each member, a home page should exist which contains (at least) the full name and the status of this member. The marking information establishes that the property must be checked only on home pages. Second 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. Third rule states that blinking text is forbidden in the whole Web site. The last rule states that, for each research project with two grants, the total project budget must be equal to the sum of the grants.

### 3.3 Simulation and partial rewriting

Partial rewriting extracts “some pieces of information” from a page, pieces them together, and then rewrites the glued term. The assembling is done by means of tree simulation, which recognizes the structure and the labeling of a given term inside a particular page of the Web site. Our notion of simulation, $\preceq$, is an adaptation of Kruskal’s *embedding* (or “syntactically simpler”) relation [5] where we ignore the usual *diving* rule\(^5\) [9].

**Definition 3.2** The *simulation* relation $\preceq \subseteq \tau(\text{Text} \cup \text{Tag}) \times \tau(\text{Text} \cup \text{Tag})$ on Web pages is the least relation satisfying the rule:

\[
f(t_1, \ldots, t_n) \preceq g(s_1, \ldots, s_n) \text{ iff } f \equiv g \text{ and } t_i \preceq s_{\pi(i)}, \text{ for } i = 1, \ldots, m, m \geq n
\]

and some injective function $\pi : \{1, \ldots, m\} \rightarrow \{1, \ldots, n\}$.

Given two Web pages $s_1$ and $s_2$, if $s_1 \preceq s_2$ we say that $s_1$ *simulates* (or *is embedded* or *recognized into*) $s_2$. We also say that $s_2$ *embeds* $s_1$. Note that, for the case when $m$ is 0 we have $c \preceq c$ for each constant symbol $c$.

Now we are ready to introduce the *partial rewrite* relation between Web page templates. Roughly speaking, given a Web specification rule $l \rightarrow r$, partial rewriting allows us to extract from, a given Web page $s$, a subpart of $s$ which is simulated by a ground instance of $l$, and to replace $s$ by a ground,

\(^5\) The *diving* rule allows one to “strike out” a part of the term at the right-hand side of the relation $\preceq$. Formally, $s \preceq f(t_1, \ldots, t_n)$, if $s \preceq t_i$, for some $i$. 198
irreducible form of an instance of $r$ w.r.t. $R$, which is computed by standard rewriting.

### 3.4 Error diagnoses

We classify the kind of errors which can be found in a Web site in terms of the different outputs delivered by our verification technique when is fed with a Web site specification.

**Definition 3.3** [correctness error] Let $\mathcal{W}$ be a Web site and $(I_M, I_N, R)$ be a Web specification. Then, the quadruple $(p,w,1,\sigma)$ is a correctness error evidence iff $p \in \mathcal{W}$, $w \in OTag(p)$, and $1\sigma$ is an instance of the left-hand side $1$ of a correctness rule belonging to $I_N$ such that $1\sigma \sqsubseteq p|w$.

Therefore, by analyzing such an evidence, we exactly know where the faulty information is located inside the Web page. We denote the set of all correctness error evidences of a Web site $\mathcal{W}$ w.r.t. a set of correctness rules $I_N$ by $E_N(\mathcal{W})$. When no confusion can arise, we just write $E_N$.

As for completeness errors, we can distinguish three classes of errors. Given a Web site $\mathcal{W}$, we have: Missing Web pages, the Web site $\mathcal{W}$ lacks one or more Web pages, Universal completeness error, the Web site $\mathcal{W}$ fails to fulfill the requirement that a piece of information must occur in all the Web pages of a given subset of $\mathcal{W}$, and Existential completeness error, the Web site $\mathcal{W}$ fails to fulfill the requirement that a piece of information must occur in some Web page of $\mathcal{W}$.

The verification methodology of [2] generates the sets of correctness and completeness error evidences $E_N$ and $E_M$ mentioned above for a given Web site w.r.t. the input Web specification. Starting from these sets, in the following section we formulate a method for fixing the errors and delivering a Web site which is correct and complete w.r.t. the intended Web specification.

### 4 Repairing a faulty Web site

Given a faulty Web site $\mathcal{W}$ and the sets of errors $E_N$ and $E_M$, our goal is to modify the given Web site by adding, changing, and removing information in order to produce a Web site that is correct and complete w.r.t. the considered Web specification. For this purpose, in correspondence with the error categories distinguished in the previous section, we introduce a catalogue of repair actions which can be applied to the faulty Web site. The primitive repair actions we consider are the following: **change**$(p,w,t)$ replaces the subterm $p|w$ in $p$ with the term $t$ and returns the modified Web page. **insert**$(p,w,t)$ modifies the term $p$ by adding the term $t$ into $p|w$ and returns the modified Web page, **add**$(p,\mathcal{W})$ adds the Web page $p$ to the Web site $\mathcal{W}$ and returns the modified Web page, **delete**$(p,t)$ deletes all the occurrences of the term $t$ in the Web page $p$ and returns the modified Web page.
We have to ensure that the data considered for insertion are safe w.r.t. the Web specification, i.e. they cannot fire any correctness rule. For this purpose, we introduce the following definition.

**Definition 4.1** Let \((I_M, I_N, R)\) be a Web specification and \(p \in \tau(\text{Text} \cup \text{Tag})\) be a Web page. Then, \(p\) is safe w.r.t. \(I_N\), iff for each \(w \in O_{\text{Tag}}(p)\) and \((l \vdash r | C) \in I_N\), either (i) there is no \(\sigma\) s.t. \(l\sigma \triangleq p_{|w};\) or (ii) \(l\sigma \triangleq p_{|w}\), but \(C\sigma\) does not hold.

### 4.1 Fixing correctness errors

Throughout this section, we will consider a given Web site \(W\), a Web specification \((I_M, I_N, R)\) and the set \(E_N \neq \emptyset\) of the correctness error evidences w.r.t. \(I_N\) for \(W\). Our goal is to modify \(W\) in order to generate a new Web site which is correct w.r.t. \((I_M, I_N, R)\). We proceed as follows: whenever a correctness error is found, we choose a possible repair action (among the different actions described below) and we execute it in order to remove the erroneous information, provided that it does not introduce any new bug.

Given \(e = (p, w, l, \sigma) \in E_N\), \(e\) can be repaired in two distinct ways: we can decide either 1) to remove the wrong content \(l\sigma\) from the Web page \(p\) (specifically, from \(p_{|w}\)), or 2) to change \(l\sigma\) into a piece of correct information. Hence, it is possible to choose between the following repair strategies.

#### 4.1.1 “Correctness through Deletion” strategy

In this case, we simply remove all the occurrences of the subterm \(p_{|w}\) of the Web page \(p\) containing the wrong information \(l\sigma\) by applying the repair action \(\text{delete}(p, p_{|w})\).\(^6\)

#### 4.1.2 “Correctness through Change” strategy

Given a correctness error \(e = (p, w, l, \sigma) \in E_N\), we replace the subterm \(p_{|w}\) of the Web page \(p\) with a new term \(t\) introduced by the user. The new term \(t\) must ensure that \(t\) does not embed subterms which might fire some correctness rule (local correctness property). Next, we have to guarantee that \(t\), within the context surrounding it, will not cause any new correctness error (global correctness property).

**Local correctness property.** This property guarantees that a change action is “locally safe”. For it we handle conditional and unconditional correctness rules separately. For conditional rules, we must look for solutions to the following problem. Let us consider the correctness error evidence \(e = (p, w, l, \sigma) \in E_N\) and the associated repair action \(\text{change}(p, w, t)\). We build the set of conditions

\[ CS_e \equiv \{ \neg C \mid \exists (1 \vdash r \mid C) \in I_N, \text{ a position } u', \text{ a substitution } \sigma \text{ s.t. } l\sigma \triangleq p_{|w,u'} \} \]

\(^6\) Note that, instead of removing the whole subterm \(p_{|w}\), it would be also possible to provide a more precise though also time-expensive implementation of the delete action which only gets rid of the part \(l\sigma\) of \(p_{|w}\) which is responsible for the correctness error.
We call $CS_e$ the constraint satisfaction problem associated with $e$. Such collection of constraints, that can be solved manually or automatically by means of an appropriate constraint solver [3], can be used to provide suitable values for the term $t$ to be inserted. We say that $CS_e$ is satisfiable iff there exists at least one assignment of values for the variables occurring in $CS_e$ that satisfies all the constraints. We denote by $Sol(CS_e)$ the set of all the assignments that verify the constraints in $CS_e$. The restriction of $Sol(CS_e)$ to the variables occurring in $\sigma$ is denoted by $Sol(CS_e)|_{\sigma}$.

Let us now consider unconditional rules. Sometimes we may need to change not only the values of the variables but also the structure of the term containing the erroneous data. In this case, it might happen that we introduce a “forbidden” structure. Therefore, in order to ensure correctness, the following structural correctness check on the structure of term $t$ must be performed only for unconditional rules.

$$\forall l \rightarrow r \in I_N, w \in O_{Tag}(t), \text{substitution } \sigma, 1 \sigma \not\sqsubseteq t|_w.$$ (1)

Roughly speaking, the structural correctness property (1) defined above ensures that no unconditional correctness rule can be triggered on $t$. This is achieved by checking that no left-hand side of an unconditional correctness rule is embedded into $t$.

**Definition 4.2** Given $e = (p, w, l, \sigma) \in E_N$ and a repair action $\text{change}(p, w, t)$, we say that $\text{change}(p, w, t)$ obeys the local correctness property iff

- for each conditional rule $(l \rightarrow r | C) \in I_N$, $C \neq \emptyset$, substitution $\sigma'$ and position $w'$, if $1\sigma' \not\sqsubseteq t|_{w'}$ then (i) $\sigma' \in Sol(CS_e)|_{\sigma'}$, when $\neg C \in CS_e$; (ii) $C\sigma'$ does not hold, when $\neg C \notin CS_e$.

- for unconditional rules, the structural correctness property (1) holds.

**Global correctness property.** Whenever we fix some wrong data by executing a repair action $\text{change}(p, w, t)$, we also need to consider $t$ within the context that surrounds it in $p$. If we don’t pay attention to such a global condition, some subtle correctness errors might arise. The definition of the Global correctness property is to consider the Local correctness property at level of the page on which the error becoming. The execution of a change action which obeys the global as well as the local correctness property, decreases the number of correctness errors of the original Web site as stated by the following proposition.

### 4.2 Fixing completeness errors

In this section, we address the problem of repairing an incomplete Web site $\hat{W}$. Without loss of generality, we assume that $\hat{W}$ is an incomplete but correct Web site w.r.t. a given Web specification $(I_M, I_N, R)$. Such an assumption will allow us to design a repair methodology which “completes” the Web site and does not introduce any incorrect information.
Let $E_M(\mathcal{W})$ be the set of completeness error evidences risen by $I_M$ for the Web site $\mathcal{W}$. Any completeness error evidence belonging to $E_M(\mathcal{W})$ can be repaired following distinct strategies and thus by applying distinct repair actions. On the one hand, we can think of adding the needed data, whenever a Web page or a piece of information in a Web page is missing. We must ensure that the execution of the chosen repair action does not introduce any new correctness/completeness error to guarantee the termination and the soundness of our methodology.

4.2.1 “Completeness through Insertion” strategy.

According to the kind of completeness error that we have to fix, we consider two distinct kinds of repair actions, namely $\text{add}(p, \mathcal{W})$ and $\text{insert}(p, w, t)$. The former action adds a new Web page $p$ to a Web site $\mathcal{W}$ and thus will be employed whenever the system has to fix a given missing Web page error. The latter allows us to add a new piece of information $t$ to (a subterm of) an incomplete Web page $p$, and therefore is suitable to repair universal as well as existential completeness errors. The insertion repair strategy works as follows.

**Missing Web page errors.** Given a missing Web page error evidence $(r, \mathcal{W})$, we fix the bug by adding a Web page $p$, which embeds the missing expression $r$, to the Web site $\mathcal{W}$. $\mathcal{W} = \mathcal{W} \cup \{ \text{add}(p, \mathcal{W}) \}$, where $r \leq p|_w$ for some $w \in O_{tag}(p)$.

**Existential completeness errors.** Given an existential completeness error evidence $(r, \{p_1, p_2, \ldots, p_n\}, \mathcal{E})$, we fix the bug by inserting a term $t$, that embeds the missing expression $r$, into an arbitrary page $p_i$, $i = 1, \ldots, n$. The position of the new piece of information $t$ in $p_i$ is typically provided by the user, who must supply a position in $p_i$ where $t$ must be attached. The $\text{insert}$ action will transform the Web site $\mathcal{W}$ in the following way: $\mathcal{W} = \mathcal{W} \setminus \{p_i\} \cup \{ \text{insert}(p_i, w, t) \}$, where $r \leq p|_w$ for some $w \in O_{tag}(p)$.

**Universal completeness errors.** Given a universal completeness error evidence $(r, \{p_1, p_2, \ldots, p_n\}, \mathcal{A})$, we fix the bug by inserting a term $t_i$, that embeds the missing expression $r$, into every Web page $p_i$, $i = 1, \ldots, n$ not embedding $r$. The position of the new piece of information $t_i$ in each $p_i$ is typically provided by the user, who must supply a position $w_i$ in $p_i$ where $t_i$ must be attached. In this case, we will execute a sequence of $\text{insert}$ actions, exactly one for each incomplete Web page $p_i$. Therefore, the Web site $\mathcal{W}$ will be transformed in the following way. For each $p_i \in \{p_1, p_2, \ldots, p_n\}$ such that $r \not\subseteq p|_{w_j}$ for each $w_j \in O_{tag}(p_i)$, $\mathcal{W} = \mathcal{W} \setminus \{p_i\} \cup \{ \text{insert}(p_i, w_i, t_i) \}$, where $r \leq p|_w$, for some $w_i \in O_{tag}(p_i)$.

**Definition 4.3** Let $(I_M, I_N, R)$ be a Web specification and $\mathcal{W}$ be a Web site w.r.t. $(I_M, I_N, R)$. Let $E_M(\mathcal{W})$ be the set of completeness error evidences of $\mathcal{W}$ w.r.t. $I_M$.

- the repair action $p_1 \equiv \text{insert}(p, w, t)$ is acceptable w.r.t. $(I_M, I_N, R)$ and $\mathcal{W}$ iff (i) $p_1$ is safe w.r.t. $(I_M, I_N, R)$; (ii) $r \leq t|_w$, $w \in O_{tag}(t)$, for some
\( e(\mathbf{r}) \in E_M(\mathcal{W}) \); (iii) if \( \mathcal{W}' \equiv \mathcal{W} \setminus \{p\} \cup \{p_1\} \), then \( E_M(\mathcal{W}') \subset E_M(\mathcal{W}) \).

- the repair action \( p_2 \equiv \text{add}(p_2, \mathcal{W}) \) is acceptable w.r.t. \((I_M, I_N, R)\) and \( \mathcal{W} \) iff
  
  (i) \( p_2 \) is safe w.r.t. \((I_M, I_N, R)\); (ii) \( \mathbf{r} \leq p_{2|w}, \ w \in O_{Tag}(p_2) \), for some \( e(\mathbf{r}) \in E_M(\mathcal{W}) \); (iii) if \( \mathcal{W}' \equiv \mathcal{W} \cup \{p_2\} \), then \( E_M(\mathcal{W}') \subset E_M(\mathcal{W}) \).

Definition 4.3 guarantees that the information which is added by \textit{insert} and \textit{add} actions is correct and does not yield any new completeness error. More precisely, the number of completeness errors decreases by effect of the execution of such repair actions.

### 4.2.2 “Completeness through Deletion” strategy.

The main idea of the deletion strategy is to remove all the information in the Web site that caused a given completeness error. The strategy is independent of the kind of completeness error we are handling. More formally, given a Web specification \((I_M, I_N, R)\), a Web site \( \mathcal{W} \) and a completeness error evidence \( e(\mathbf{r}) \), the Web site \( \mathcal{W} \) will change in the following way. For each \( t_1 \rightarrow t_2 \rightarrow \ldots \rightarrow \mathbf{r} \), where \( t_i \leq p_{|w}, \ w \in O_{Tag}(p), p \in \mathcal{W} \)

\[
\mathcal{W} \equiv \{p \in \mathcal{W} \mid t_i \not\leq p_{|w}, \forall w \in O_{Tag}(p), i = 1, \ldots, n\} \cup \\
\{\text{delete}(p, t_i) \mid p \in \mathcal{W}, t_i \leq p_{|w}, w \in O_{Tag}(p), i = 1, \ldots, n\}
\]

As in the case of the insertion strategy, we have to take care about the effects of the execution of the repair actions. More precisely, we do not want the execution of any \textit{delete} action to introduce new completeness errors. For this purpose, we consider the following notion of acceptable delete action.

\textbf{Definition 4.4} Let \((I_M, I_N, R)\) be a Web specification and \( \mathcal{W} \) be a Web site w.r.t. \((I_M, I_N, R)\). Let \( E_M(\mathcal{W}) \) be the set of completeness error evidences of \( \mathcal{W} \) w.r.t. \( I_M(\mathcal{W}) \). The repair action \( p_1 \equiv \text{delete}(p, t) \) is acceptable w.r.t. \((I_M, I_N, R)\) and \( \mathcal{W} \) iff if \( \mathcal{W}' \equiv \mathcal{W} \setminus \{p\} \cup \{p_1\} \), then \( E_M(\mathcal{W}') \subset E_M(\mathcal{W}) \).

### 5 Conclusions

Maintaining contents of Web sites is an open and urgent problem since outdated, incorrect and incomplete information is becoming very frequent in the World Wide Web. In this paper, we presented a semi-automatic methodology for repairing Web sites which has a number of advantages over other potential approaches (and hence can be used as a useful complement to them): 1) in contrast to the active database Web management techniques, we are able to predict whether a repair action can cause new errors to appear and assist the user in reformulating the action; 2) by solving the constraint satisfaction problem associated to the conditions of the Web specification rules, we are also able to aid users to fix erroneous information by suggesting ranges of correct values; 3) our methodology smoothly integrates on top of existing
rewriting-based web verification frameworks such as [2,1], which offer the expressiveness and computational power of functions and allow one to avoid the encumbrances of DTDs and XML rule languages. The basic methodology has been partially implemented in the preliminary prototype GVerdi-R (Graphical Verification and Rewriting for Debugger Internet sites), publicly available together with a set of examples at http://www.dsic.upv.es/users/elp/GVerdi and which is written in Haskell (GHC v6.2.2).

Let us conclude by discussing further works. For the benefit of the user who prefers not to express himself in a formal language, we are currently working on a graphical notation for rules and for the repair actions, which will be automatically translated into our formalism. Moreover, to increase the level of automation of our repair method, we are working on possible correction strategies which minimize both the amount of information to be changed and the number of repair actions to be executed.

References