

An Abstract Generic Framework for Web Site Verification *

M. Alpuente P. Ojeda D. Romero
 Technical University of Valencia
 Camino de Vera s/n, Apdo. 22012,
 46071 Valencia, Spain.
 {alpuente,pojeda,dromero}@dsic.upv.es

D. Ballis
 University of Udine
 Via delle Scienze 206,
 33100 Udine, Italy.
 demis.ballis@dimi.uniud.it

M. Falaschi
 University of Siena
 Pian dei Mantellini 44,
 53100 Siena, Italy.
 falaschi@dimi.uniud.it

Abstract

In this paper, we present an abstract framework for Web site verification which improves the performance of a previous, rewriting-based Web verification methodology. The approximated framework is formalized as a source-to-source transformation which is parametric w.r.t. the chosen abstraction. This transformation significantly reduces the size of the Web documents by dropping or merging contents that do not influence the properties to be checked. This allows us to reuse all verification facilities of the previous system WebVerdi-M to efficiently analyze Web sites. In order to ensure that the verified properties are not affected by the abstraction, we develop a methodology which derives the abstraction of Web sites from their Web specification. An experimental implementation shows a huge speedup w.r.t. a previous methodology which did not use this transformation.

1. Introduction

Despite the exponential WWW growth and the success of the Semantic Web, there is limited support today to specifying, verifying and repairing Web sites at a semantic level. Some sophisticated Web site management tools have been recently proposed that provide helpful facilities, including active rules that automatically fire repair actions throughout the rich navigational structure of Web sites. Unfortunately, these tools are mostly focused towards syntactic checking (see [3, 4, 6] for a wider discussion).

A rewriting-based approach to Web-site verification and repair was developed in [3, 7]. The methodology applies to static-HTML/XML Web sites and can discover flaws in Web sites that are not addressed by classical tools [2, 15, 21] as these mainly focus on modeling navigational aspects and

user interaction; see [1]. Our technique is particularly suitable to analyze *collaborative* Web sites, (that is, sites where several users may freely change/remove data). In this scenario, keeping the static web contents complete and correct is obviously not trivial and requires advanced verification capabilities which are naturally supported by our methodology. In a nutshell, our framework comes with a Web specification language for defining correctness and completeness conditions on Web sites. Then, a rewriting-based verification technique is applied to recognize forbidden/incorrect patterns and incomplete/missing Web pages. This is done by means of a novel technique, called *partial rewriting*, in which the traditional pattern matching mechanism is replaced by a suitable technique based on an *homeomorphic embedding* relation for recognizing patterns inside semistructured documents.

The verification methodology of [3] is implemented in the prototype WebVerdi-M (Web VErification and Rewriting for Debugging Internet sites with Maude) [4], written in Maude [10]. For correctness checking, it shows impressive performance thanks to the Associativity-Commutativity (AC) pattern matching and metalevel features supported by Maude (for instance, verifying correctness over a 10Mb XML document with 302000 nodes takes less than 13 seconds). Both resource allocation and elapsed time scale linearly. Unfortunately, for the verification of completeness, a (finite) fixpoint computation is typically needed which leads to unsatisfactory performance, and the verification tool is only able to efficiently process XML documents smaller than 1Mb.

In this paper, we develop an abstract approach to Web site verification which makes use of an approximation technique based on abstract interpretation [12, 13] that greatly improves on previous performance. We also ascertain the conditions which ensure the correctness of the approximation, so that the resulting abstract rewriting engine safely supports accurate Web site verification. Since our framework is parametric w.r.t. the considered abstraction, we precisely characterize the conditions which allow us to ensure the correctness of the abstraction, which is implemented

*This work has been partially supported by the EU (FEDER) and the Spanish MEC, under grants TIN2004-7943-C04-01 and TIN2007-68093-C02-02, the Generalitat Valenciana under grant GV06/285, and Integrated Action Hispano-Alemana HA2006-0007, and progetto FIRB, Internazionalizzazione 2004, n. RBIN04M8S8. Pedro Ojeda is also supported by the Generalitat Valenciana under FPI grant BFPI/2007/076.

by a source-to-source transformation of concrete Web sites and Web specifications into abstract ones. Thanks to this source-to-source approximation scheme, all facilities supported by our previous verification system are straightforwardly adapted and reused with very little effort. Our abstract verification methodology can be seen as a step forward towards more sophisticated management of dynamic components of Web sites, which can generate a potentially infinite number of Web pages and thus cannot be handled by the more standard methodology in [3]. An extended version of this work can be found in [5].

Related Work In the literature, abstract interpretation frameworks have been scarcely applied to analyse Web sites. In [18] an abstract approach is developed which allows one to analyse the communication protocols of a particular distributed system with the aim of enforcing a correct global behavior of the system. [17] uses abstract interpretation for secret property verification: the methodology applies Input/Output abstract set descriptions to finite state machines in order to validate cryptographic protocols implementing secure Web transactions.

To the best of our knowledge, this work develops the first methodology based on abstract interpretation techniques which is general enough to support the verification of static as well as dynamic aspects of Web sites. Our inspiration comes from the area of approximating (XML) query answering [8, 22], where XML queries are executed on compressed versions of XML data (i.e., document synopses) in order to obtain fast, albeit approximate, answers.

Plan of the Paper The paper is organized as follows. Section 2 recalls some standard notions, and introduces Web site descriptions. In Section 3, we briefly recall the Web verification methodology of [3]. Section 4 formalizes our technique for abstract Web site verification and demonstrates its soundness. Experiments with a prototypical implementation of our method are described in Section 5. Finally, Section 6 concludes.

2. Preliminaries

In this section, we briefly recall the essential notions and terminology used in this paper. We call a finite set of symbols *alphabet*. By \mathcal{V} we denote a countably infinite set of variables and Σ denotes a set of *function symbols* (also called *operators*), or *signature*. We consider varyadic signatures as in [14] (i.e., signatures in which symbols do not have a fixed arity).

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. The empty sequence Λ denotes the root position. Given $S \subseteq \Sigma \cup \mathcal{V}$, $O_S(t)$ denotes the set of positions of a term t that are rooted by symbols in S . $t|_u$ is the subterm at the position u of t . $t[r]_u$ is the term t with the subterm rooted at the position u replaced by r . Given a term t , we say that t is *ground*, if no variable occurs

in t . $\tau(\Sigma, \mathcal{V})$ and $\tau(\Sigma)$ denote the *non-ground term algebra* built on $\Sigma \cup \mathcal{V}$ and the *term algebra* built on Σ , respectively.

Syntactic equality between objects is represented by \equiv . Given a set S , sequences of elements of S are built with constructors $\epsilon :: S^*$ (empty sequence) and $\cdot :: S \times S^* \rightarrow S^*$.

A *substitution* $\sigma \equiv \{X_1/t_1, \dots, X_n/t_n\}$ 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) $X_i\sigma = t_i$, $i = 1, \dots, n$, and (iii) $X\sigma = X$, for all $X \in \mathcal{V} \setminus \{X_1, \dots, X_n\}$. An *instance* of a term t is defined as $t\sigma$, where σ is a substitution. By $Var(s)$ we denote the set of variables occurring in the syntactic object s .

2.1. Web Site Description

Let us consider two alphabets T and $\mathcal{T}ag$. We denote the set T^* by $\mathcal{T}ext$. An object $t \in \mathcal{T}ag$ is called *tag element*, while an element $w \in \mathcal{T}ext$ is called *text element*. Since Web pages are provided with a tree-like structure, they can be straightforwardly translated into ordinary terms of the term algebra $\tau(\mathcal{T}ext \cup \mathcal{T}ag)$.

Note that XML/XHTML tag attributes can be considered as common tagged elements, and hence translated in the same way. In the following, we also consider terms of the non-ground term algebra $\tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V})$, which may contain variables. Elements of this set are called *Web page templates*. In our methodology, Web page templates are used for specifying erroneous/incorrect patterns which may be recognized in the Web pages.

In order to describe a Web site, we use the formulation given in [20]. We use an alphabet \mathcal{P} to give names to Web pages and to express the different transitions between pages.

Definition 2.1 (immediate successors) *The immediate successors relation for a given Web page p is defined by $\rightarrow_p = \{(p, p') \subseteq \mathcal{P} \times \mathcal{P} \mid p' \text{ is directly accessible from } p\}$.*

Definition 2.1 establishes a relationship between the page p and its immediate successors (i.e., the pages p_1, \dots, p_n that p points to by means of hyperlinks). We will use the associated computational relations $\rightarrow_p, \rightarrow_p^+,$ etc., to describe the dynamic behavior of a Web site. For instance, the reachability of a given Web page p' from another page p can be expressed as $p \rightarrow_p^* p'$.

Definition 2.2 (Web site) *A Web site is defined as a set of reachable Web pages from an initial Web page, and is denoted by*

$$W = \{p_1, \dots, p_n\}, \text{ s.t. } \begin{array}{l} \exists i, 1 \leq i \leq n, \\ \forall j, 1 \leq j \leq n, p_i \rightarrow_W^* p_j \end{array}$$

Definition 2.2 formalizes the idea that a Web site has an initial Web page which allows one to visit the whole Web site. Note that there may exist several initial Web pages of a given Web site.

Example 2.3 *The algebraic description of a simple Web site modeling an on-line auction system is shown in Figure 1. It contains information regarding open and closed auctions, auctioned items, and registered users.*

3. Rewriting-based Web Verification

In this section, we briefly recall the formal verification methodology proposed in [3], which allows us to detect forbidden/erroneous contents as well as missing information in a Web site.

3.1. The Web specification language

A Web specification is a triple (R, I_N, I_M) , where R , I_N , and I_M are a finite set of rules. The set R contains the definition of some auxiliary functions which the user would like to provide, such as string processing, arithmetic, boolean operators, etc. R is formalized as a term rewriting system, which is handled by standard rewriting [23].

The second set I_N describes constraints for detecting erroneous Web pages (*correctness rules*).

The third set of rules I_M specifies some conditions signaling incomplete/missing Web pages (*completeness rules*). A completeness rule is defined as $l \rightarrow r \langle q \rangle$ where l 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.

In the following we restrict ourselves to completeness verification (see [3] for a description of the full methodology).

Example 3.1 *A Web specification for the on-line auction system of Example 2.3 is also shown in Figure 1. It contains four completeness rules. The first (existential) rule formalizes the following property: if there is a Web page with the information about the seller of an open-auction, then such a seller must be registered. The second rule also states an existential property: if there is an auctioned item that is listed in two or more categories, then at least two of these categories must be “unit” and “pack”. The third rule formalizes the following universal property: for each client registered in the Web site, a page must exist containing his name. The last rule states that, for every item that is sold, a closed auction associated to the item must exist.*

3.2. Homeomorphic embedding 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 a single embedding relation, which allow us to verify whether

a given Web page template is somehow “enclosed” within another one and thus replaces the traditional pattern matching [16].

We give a definition of homeomorphic embedding, \trianglelefteq , which is an adaptation of the one proposed in [19], where (i) a simpler treatment of the variables is considered, (ii) function symbols with variable arities are allowed, (iii) the relative positions of the arguments of terms are ignored (i.e. $f(a, b)$ is not distinguishable from $f(b, a)$), and (iv) we ignore the usual *diving* rule¹ [19].

Definition 3.2 (homeomorphic embedding) *The homeomorphic embedding relation*

$$\trianglelefteq \subseteq \tau(\text{Text} \cup \text{Tag}, \mathcal{V}) \times \tau(\text{Text} \cup \text{Tag}, \mathcal{V})$$

on Web page templates is the least relation satisfying:

1. $X \trianglelefteq t$, for all $X \in \mathcal{V}$ and $t \in \tau(\text{Text} \cup \text{Tag}, \mathcal{V})$.
2. $f(t_1, \dots, t_m) \trianglelefteq g(s_1, \dots, s_n)$ iff $f \equiv g$ and $t_i \trianglelefteq s_{\pi(i)}$, for $i = 1, \dots, m$, and injective function $\pi : \{1, \dots, m\} \rightarrow \{1, \dots, n\}$.

Whenever $s \trianglelefteq t$, we say that t embeds s (or s is embedded or “recognized” in t). The intuition behind the above definition is that the structure of the template s appears within the specific Web data term t .

Now we are ready to introduce the partial rewrite relation between Web page templates. W.l.o.g., we disregard conditions and/or quantifiers from the Web specification rules. 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 then replace s by a reduced, ground instance of r . Let $s, t \in \tau(\text{Text} \cup \text{Tag}, \mathcal{V})$. Then, s partially rewrites to t via rule $l \rightarrow r$ and substitution σ iff there exists a position $u \in O_{\text{Tag}}(s)$ such that (i) $l\sigma \trianglelefteq s|_u$, and (ii) $t = \text{Reduce}(r\sigma, R)$, where function $\text{Reduce}(x, R)$ computes, by standard term rewriting, the irreducible form of x in R . Note that the context of the selected reducible expression $s|_u$ is disregarded after each partial rewrite step. By notation $s \rightarrow_I t$, we denote that s is partially rewritten to t using a rule belonging to the set I .

3.3. Web verification methodology

Roughly speaking, our verification methodology works as follows. We apply a partial rewriting step to p , that is, we replace the Web page p embedding $l\sigma$ with the instantiated right-hand side $r\sigma$, in symbols $p \rightarrow r\sigma$. Each term $r\sigma$ generated by partial rewriting is considered a *requirement*. From the requirements computed so far, new requirements

¹The diving rule allows one to “strike out” a part of the term at the right-hand side of the relation \trianglelefteq . Formally, $s \trianglelefteq f(t_1, \dots, t_n)$, if $s \trianglelefteq t_i$, for some i .

Web site $W = \{p_1, p_2, p_3, p_4, p_5\}$, where

p_1) list-items(item(id(ite0),name(racket),state(sold), description(Wilson tennis racket), incategories(category(cat1))), item(id(ite1),name(shirt),state(available), description(men's t-shirts), incategories(category(cat1), category(cat2))), item(id(ite2),name(shoes),state(sold), description(women's shoes), incategories(category(cat0), category(cat2))))	p_2) list-categories(pack(category(cat0), unit(category(cat1), category(cat2)))	p_3) people(person(id(per0), name(Eliyahu), email(eliyahu@cas.cz)), person(id(per1), name(Melski), email(melski@cabo.com)), person(id(per2), name(Conte), email(conte@forth.gr)), person(id(per3))
p_4) open-auctions(open-auction(id(open-auction0), item(ite0), initial(48.51), reserve(77.5), bidder(person(per0)), seller(person(per1)))	p_5) closed-auctions(closed-auction(seller(person(per1)), buyer(person(per0)), item(ite0), price(77.5)), closed-auction(seller(person(per5)), buyer(person(per0)), item(ite2), price(45.2))	

Web specification (I_N, I_M, R) , where $I_N = \{\}$ and $I_M = \{r_1, r_2, r_3, r_4\}$,
 r_1) open-auction(seller(person(X))) \rightarrow $\#$ person($\#$ id(X)) $\langle E \rangle$
 r_2) list-items(item(incategories(X,Y))) \rightarrow $\#$ list-categories(pack(X),unit(Y)) $\langle E \rangle$
 r_3) people(person(id(X))) \rightarrow $\#$ people($\#$ person($\#$ id(X)),name) $\langle A \rangle$
 r_4) list-item(item(id(X),state(sold))) \rightarrow $\#$ closed-auction(item(X)) $\langle E \rangle$

Figure 1. Web site and Web specification for an on-line auction system.

are generated by means of a fixpoint computation by applying further partial rewriting steps. Then, by a new homeomorphic embedding test, we check whether the given requirement $r\sigma$ is recognized within some page of the considered Web site. If the test fails, a completeness error is signalled.

When navigating a Web site, it is common to find a number of pages that have a similar structure but different contents. This happens very often when pages are dynamically generated by some script which extracts contents from a database (e.g. in Amazon's Web site). In the following we develop an abstract methodology which derives an approximation of Web sites from the considered Web specifications.

4. Abstract Web site verification

The basic idea of abstract interpretation [12, 13] is to infer information from programs by interpreting ("running") them using abstract data rather than concrete ones, thus obtaining safe approximations of the programs. The "concrete" data and operators are replaced by corresponding "abstract" (approximated) data and operators. The "answers" obtained by running the program in the domain are proven sound by exploiting the correspondence between the abstract and concrete domain.

In this work, we are particularly interested in the abstraction of completeness process, because in the previous version the needed fixpoint computation leads to unsatisfactory performance (see [4]).

We want to formalize the abstraction as a source-to-source transformation which translates Web documents and Web specification rules into constructions of the very same languages. In this way, the domain of abstract terms \mathcal{D}^α is equal to the domain of concrete terms \mathcal{D} .

Let us introduce the definition of term abstraction α .

Definition 4.1 (term abstraction α) Let $atext :: Tag^* \times Text \rightarrow Text$ be a text abstraction function.

$$\alpha :: \tau(Text \cup Tag, \mathcal{V}) \rightarrow \tau(Text \cup Tag, \mathcal{V})$$

$$\alpha(t) = \hat{\alpha}(\epsilon, t)$$

where the auxiliary function $\hat{\alpha}$ is given by

$$\hat{\alpha} :: Tag^* \times \tau(Text \cup Tag, \mathcal{V}) \rightarrow \tau(Text \cup Tag, \mathcal{V})$$

$$\hat{\alpha}(-, x) = x, \text{ if } x \in \mathcal{V}$$

$$\hat{\alpha}(c, f(t_1, \dots, t_n)) = f(\hat{\alpha}(c.f, t_1), \dots, \hat{\alpha}(c.f, t_n)), \text{ if } f \in Tag$$

$$\hat{\alpha}(c, w) = atext(c, w), \text{ if } w \in Text$$

Particularly, the reader may notice that elements of $Text$ are abstracted by taking into account the chain of tags under which a particular piece of text appears. This is formalized by means of the text abstraction function

$$atext :: Tag^* \times Text \rightarrow Text$$

which is left undefined and is actually the formal parameter of the definition.

The text abstraction function should be conveniently fixed in order to tune the abstraction for each particular domain. For instance, in the case where no tag distinction is needed, each element in $text$ could be simply replaced by some abstract fresh, constant symbol d .

In order to achieve correctness of the abstraction, we restrict our interest to text abstraction functions $atext$ which distinguish those pieces of text that are observed by the Web specification rules and then potentially affect the result of the verification.

The auxiliary function $gen_emb_tt_{\mathcal{J}}(c, t)$ allows us to know whether a sequence of tags c (with leaf t) is recognized within some rule of the Web specification \mathcal{J} . This allows us to determine whether a term within a Web page needs to be carefully considered.

Now text abstraction functions are required to obey the following correctness condition w.r.t. W .

Definition 4.2 (correctness condition w.r.t. W) Let W be a Web site and \mathcal{J} be a Web specification. Let $s, t \in \text{Text}$ be any two pieces of text in W . For every $c \in \text{Tag}^*$ such that $\text{gen_emb_tt}_{\mathcal{J}}(c, s) \equiv \text{True}$ and $\text{gen_emb_tt}_{\mathcal{J}}(c, t) \equiv \text{True}$, the text abstraction function atext satisfies

$$s \neq t \Rightarrow \text{atext}(c, s) \neq \text{atext}(c, t)$$

The condition above formalizes the idea that, whenever two pieces of text are indistinguishable in the abstract domain, then they are also indistinguishable in the concrete domain.

4.1. Abstract Web specification

The abstraction of the completeness Web specification rules is simply based on abstracting the terms occurring in the left-hand and right-hand sides of the rules.

Definition 4.3 (abstract specification rule)

Let $\alpha :: \tau(\text{Text} \cup \text{Tag}, \mathcal{V}) \rightarrow \tau(\text{Text} \cup \text{Tag}, \mathcal{V})$ be a term abstraction function with text abstraction function $\text{atext} :: \text{Tag}^* \times \text{Text} \rightarrow \text{Text}$. Let $\text{rl}_M \equiv l \rightarrow r \langle q \rangle$ be a completeness rule. We denote by rl_M^α the abstraction of rl_M , where $\text{rl}_M^\alpha \equiv \alpha(l) \rightarrow \alpha(r) \langle q \rangle$.

Example 4.4 Consider the completeness rule r_4 of Example 3.1. By fixing the text abstraction function $\text{atext}(c, x) = \text{last}(x)$ where $\text{last}(w)$ returns the last element of the sequence w , the computed abstract completeness rule $\alpha(r_4)$ is $\text{list-item}(\text{item}(\text{id}(X), \text{state}(d)) \rightarrow \# \text{closed-auction}(\text{item}(X)) \langle E \rangle$.

When no confusion can arise, we just write $\text{rl}_M^\alpha \equiv l^\alpha \rightarrow r^\alpha \langle q \rangle$. The Web specification (I_N, I_M, R) is lifted to $(I_N^\alpha, I_M^\alpha, R)$ element-wise².

4.2. Abstract Web site

Let us introduce a compression function for terms which is used as a preprocess prior to the abstraction of a given Web site.

4.2.1. Web Compression pre-processing

Given $f(t_1, \dots, t_n) \in \tau(\text{Text} \cup \text{Tag})$, the function $\text{COMPRESS}(f(t_1, \dots, t_n), I_M)$ packs together those subterms of $f(t_1, \dots, t_n)$ which are at the same depth and rooted by the same root symbol while ensuring that the arity of f after the transformation is not smaller than maximal arity of f in I_M . This condition of maximal arity of f in I_M ensures that a partial rewrite step on an abstract term is always enabled, whenever the corresponding partial rewrite step can be executed in the concrete domain (see [5] for more detail).

Now we are ready to formalize our notion of Web site approximation.

²Abstraction of correctness rules can be found in [5].

4.2.2. Web site abstraction

In order to approximate a Web site, we start from an initial Web page and recursively apply the successor relation (\rightarrow), while implementing a simple *depth-first* search (DFS) [11].

Definition 4.5 (Abstract Web Site) Let W be a Web site, p be an initial page of W , and (I_N, I_M, R) be a Web specification. Then, the abstraction of W is defined by:

$$\alpha(W) = \text{DFS}(p, \emptyset, I_M)$$

Where function DFS is given in Algorithm 1.

Algorithm 1 Web site abstraction.

Input:

$p :: \tau(\text{Text} \cup \text{Tag}, \mathcal{V})$
 $W^\alpha :: \text{set}(\tau(\text{Text} \cup \text{Tag}, \mathcal{V}))$
 I_M a set of completeness rules

Output:

$W^\alpha = \text{set}(\tau(\text{Text} \cup \text{Tag}, \mathcal{V}))$
1: **function** DFS (p, W^α, I_M)
2: $p^\alpha \leftarrow \text{COMPRESS}(\alpha(p), I_M)$
3: $W^\alpha \leftarrow W^\alpha \cup \{p^\alpha\}$
4: **for all** i s.t. $(p, p_i) \in \rightarrow_p$ **and**
 $\text{COMPRESS}(\alpha(p_i), I_M) \notin W^\alpha$ **do**
5: $W^\alpha \leftarrow \text{DFS}(p_i, W^\alpha)$
6: **end for**
7: $\leftarrow W^\alpha$
8: **end function**

4.3. Abstract verification soundness

Informally, our abstract verification methodology applies to the considered abstract descriptions of the Web site and Web specification. Given a Web specification (I_N, I_M, R) and a Web site W , we first generate the corresponding abstractions $(I_N^\alpha, I_M^\alpha, R)$ and W^α . Then — since we consider a source to source transformation — we apply our original verification algorithm [3] to analyse W^α w.r.t. $(I_N^\alpha, I_M^\alpha, R)$. We call *abstract error*, each error which is detected in W^α using $(I_N^\alpha, I_M^\alpha, R)$ by the verification methodology.

In order to guarantee the soundness of the abstract diagnosis, we have to ensure that, when fed with the abstracted data, the partial rewriting relation, \rightarrow , correctly approximates the behavior of the partial rewriting relation over the corresponding concrete representation. In the following, we present some results which state the soundness of our abstract representation. The extended version of this work (see [5]) contains the proofs of these results. First of all we introduce the notion of *abstract embedding*, which is used to establish a relation between concrete and abstract terms.

Definition 4.6 (abstract embedding) The abstract embedding relation

$$\leq^\# \subseteq \tau(\text{Text} \cup \text{Tag}, \mathcal{V}) \times \tau(\text{Text} \cup \text{Tag}, \mathcal{V})$$

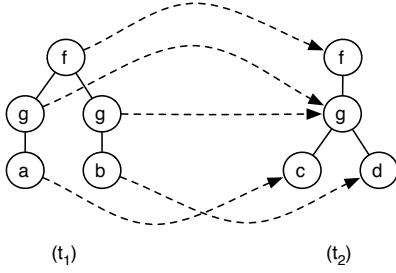


Figure 2. Abstract embedding

w.r.t. a function $atext :: Tag^* \times Text \rightarrow Text$ on Web page templates is the least relation satisfying the rules:

1. $X \triangleq^{\#} t$, for all $X \in \mathcal{V}$ and $t \in \tau(Text \cup Tag, \mathcal{V})$.
2. $f(t_1, \dots, t_m) \triangleq^{\#} g(s_1, \dots, s_n)$ iff $f \equiv g$ and $t_i \triangleq^{\#} s_{\pi(i)}$, for $i = 1, \dots, m$, and some total function $\pi :: \{1, \dots, m\} \rightarrow \{1, \dots, n\}$.
3. $c \triangleq^{\#} c'$ iff $c' \equiv atext(x, c)$ for some $x \in Tag^*$.

By using Definition 4.6, we are able to map any concrete path of a concrete term into a path of an abstract term: the structure and the labeling of t are represented in a compressed and suitable relabeled version of t in which many paths of t are mapped to one shared path of the abstract description as stated by the following proposition.

Proposition 4.7 Let $t \in \tau(Text \cup Tag, \mathcal{V})$. Let $\alpha :: \tau(Text \cup Tag, \mathcal{V}) \rightarrow \tau(Text \cup Tag, \mathcal{V})$ be a term abstraction function whose text abstraction function is $atext :: Tag^* \times Text \rightarrow Text$. Then, $t \triangleq^{\#} \alpha(t)$ w.r.t. $atext$.

Roughly speaking, Proposition 4.7 says that $\alpha(t)$ safely approximates the (concrete) term t .

Example 4.8 Consider the terms t_1 and t_2 of Figure 2. Let $atext :: Tag^* \times Text \rightarrow Text$ be defined as $\{(f.g, a) \mapsto c, (f.g, b) \mapsto d\}$. Assume that there exists a Web specification in which the maximal arity of g equals to 1, so that the compression of the t_1 's nodes labeled with g is enabled. Then, $\alpha(t_1) \equiv t_2$ and $t_2 \triangleq^{\#} \alpha(t_1)$ w.r.t. function $atext$.

In the following, we demonstrate that whenever a partial rewrite step $t_1 \rightarrow t_2$ is executed in the concrete domain, a partial rewrite step over the abstract counterpart is enabled, in symbols $\alpha(t_1) \rightarrow t'_2$, such that t_2 still simulates the obtained abstract term t'_2 w.r.t. $\triangleq^{\#}$.

Proposition 4.9 Let $\alpha :: \tau(Text \cup Tag, \mathcal{V}) \rightarrow \tau(Text \cup Tag, \mathcal{V})$ be a term abstraction function with text abstraction function $atext :: Tag^* \times Text \rightarrow Text$. Let $\mathcal{J} \equiv (I_N, I_M, R)$ be a Web specification, and $\mathcal{J}^\alpha \equiv (I_N^\alpha, I_M^\alpha, R)$ be the abstract version of \mathcal{J} .

If $t_0 \rightarrow_{I_M} t_1 \rightarrow_{I_M} \dots \rightarrow_{I_M} t_n$, $n \geq 0$, then

1. $\alpha(t_0) \rightarrow_{I_M^\alpha} t'_1 \rightarrow_{I_M^\alpha} \dots \rightarrow_{I_M^\alpha} t'_n$;
2. $t_n \triangleq^{\#} t'_n$ w.r.t. $atext$

Given an (abstract) partial rewrite sequence $\mathcal{S}^\alpha \equiv \alpha(t_1) \rightarrow_{I_M^\alpha} t'_2 \rightarrow_{I_M^\alpha} \dots \rightarrow_{I_M^\alpha} t'_n$, we call abstract completeness requirement any term appearing in \mathcal{S}^α . Now, the concrete verification methodology works as follows: first, we compute the concrete completeness requirements and, then, we check whether such requirements are fulfilled in the considered Web site. As explained in Section 3.3, a completeness requirement r is a term which occurs in a partial rewrite sequence of the form $p \equiv t_0 \rightarrow t_1 \rightarrow t_2 \dots \rightarrow t_n \equiv r$, where $p \in \tau(Text \cup Tag)$ is a Web page of the Web site W and $r \in \tau(Text \cup Tag)$ is the computed (completeness) requirement. Our novel abstract methodology exploits Proposition 4.9 to avoid the computation of concrete requirements, since they are safely approximated by their abstract descriptions.

The fact that any concrete completeness requirement is safely approximated by an abstract completeness requirement ensures that the abstract verification is safe, that is, whenever an abstract requirement is fulfilled in the abstract Web site, each concrete representation is fulfilled in the concrete domain. This allows us to conclude the absence of concrete errors in the case when no abstract errors are detected. Note that —whenever we detect an abstract completeness error— we are not able to guarantee the presence of a concrete completeness error. This is mainly due to the fact that the abstraction can enable partial rewriting steps over abstract descriptions which are not possible in the concrete domain. Thus, there might be an abstract requirement which does not correspond to any concrete requirement.

5. Implementation

An experimental implementation α Verdi of the abstract framework proposed in this paper has been developed and compared to the previous Verdi implementation for the realistic test cases given in [4]. Table 1 shows some of the results we obtained for the simulation of the Web specification rules for an on-line auction system in five different, randomly generated XML documents. Specifically, we tuned the generator `xmlgen` (available within the XMark project [9]), for scaling factors from 0.01 to 0.1 to produce XML documents whose size ranges from 1Mb (corresponding to an XML tree of about 30 thousand nodes) to 10Mb (corresponding to an XML tree of about 301 thousand nodes).

Nodes	Mb	Time		
		Verdi	Abstraction	
			App	α Verdi
30 th	1	165.34 s	11 s	0.92 s
90 th	3	1,768.65 s	154 s	3.01 s
150 th	5	4,712.39 s	732 s	52.45 s
241 th	8	12,503.85 s	5,330 s	186.22 s
301 th	10	21,208.28 s	8,132 s	285.51 s

Table 1. Verdi and α Verdi Benchmarks

The results shown in Table 1 were obtained on a personal computer equipped with 1Gb of RAM memory, 40Gb hard disk and a Pentium Centrino CPU clocked at 1.75 GHz running Ubuntu Linux 7.04.

Column *Verdi* shows the runtime of the original *Verdi* tool. Column *App* shows the time used for the approximation of the Web site w.r.t. the corresponding abstract Web specification. Finally, column α *Verdi* shows the execution time of the abstract verification tool α *Verdi*.

The preliminary results that we have obtained demonstrate a huge speedup w.r.t. our previous methodology. At the same time, the abstraction times are affordable given the complexity and size of the involved data sets: less than 5 minutes for the largest benchmark (10 Mb), with a very reduced space budget. We note that the original *Verdi* implementation was only able to process efficiently XML documents whose size was not bigger than 1Mb.

6. Conclusion

Web developing tends to create data that exhibit many commonalities which often result in extremely inefficient Web site verification models and methodologies. This paper describes a novel abstract methodology for Web sites analysis and verification which offsets the high execution costs of analyzing complex Web documents. The framework is formalized as a source-to-source transformation which is parametric w.r.t. the abstraction and translates the Web documents and their specifications into constructions of the very same languages, so that an efficient implementation can be easily derived with very little effort. The key idea for the abstraction is to exploit the sub-structure similarity that is commonly found in HTML/XML documents. Note that no automatic abstraction refinement is needed because no relevant parts are lost due to abstraction.

To conclude, by the approximation scheme formalized so far, we are able to apply the original verification framework to abstract data, providing an extremely efficient analysis which is able to ensure the absence of completeness errors in the concrete descriptions quickly, saving time to the user.

References

- [1] M. H. Alalfi, J. R. Cordy, and T. R. Dean. A Survey of Analysis Models and Methods in Website Verification and Testing. In *Proc. 7th Int'l Conf. on Web Engineering (ICWE 2007)*, volume 4607 of *LNCS*, pages 306–311, 2007.
- [2] L. d. Alfaro. Model checking the world wide web. In *Proc. 13th Int'l. Conf. Comp. Aided Verification (CAV 2001)*, Paris, France. V.2102. pp 337-349, LNCS, 2001.
- [3] M. Alpuente, D. Ballis, and M. Falaschi. Rule-based Verification of Web Sites. *Software Tools for Technology Transfer*, 8:565–585, 2006.
- [4] M. Alpuente, D. Ballis, M. Falaschi, P. Ojeda, and D. Romero. A Fast Algebraic Web Verification Service. In *Proc. of First Int'l Conf. on Web Reasoning and Rule Systems (RR 2007)*. V.4524, of *LNCS*, pp 239-248, 2007.
- [5] M. Alpuente, D. Ballis, M. Falaschi, P. Ojeda, and D. Romero. Abstract Web Site Verification in WebVerdiM. Technical Report DSIC-II/19/07, DSIC-UPV, 2007.
- [6] M. Alpuente, D. Ballis, M. Falaschi, and D. Romero. A Semi-automatic Methodology for Repairing Faulty Web Sites. In *Proc. of the 4th IEEE Int'l Conference on Software Engineering and Formal Methods (SEFM'06)*, pages 31–40. IEEE Computer Society Press, 2006.
- [7] D. Ballis and J. G. Vivó. A Rule-based System for Web Site Verification. In *Proc. of 1st Int'l Workshop on Automated Specification and Verification of Web Sites (WWV'05)*, volume 157(2). ENTCS, Elsevier, 2005.
- [8] P. Buneman, M. Grohe, and C. Koch. Path Queries on Compressed XML. In *Proc. of the 29th Int'l Conf. on Very Large Data Bases (VLDB'03)*, pages 141–152, 2003.
- [9] Centrum voor Wiskunde en Informatica. XMark – an XML Benchmark Project, 2001. Available at: <http://monetdb.cwi.nl/xml/>.
- [10] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. The maude 2.0 system. In *Rewriting Techniques and Applications (RTA 2003)*, number 2706 in *LNCS*, pages 76–87, 2003.
- [11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, USA, 2nd edition, 2001.
- [12] P. Cousot and R. Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Appr. of Fixpoints. In *POPL*, pages 238–252, 1977.
- [13] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *POPL*, pages 269–282, 1979.
- [14] N. Dershowitz and D. Plaisted. Rewriting. *Handbook of Automated Reasoning*, 1:535–610, 2001.
- [15] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. Verifying Integrity Constraints on Web Sites. In *Proc. of 16th Int'l Joint Conf. on Artificial Intelligence (IJCAI'99)*, pp 614–619, 1999.
- [16] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing Simulations on Finite and Infinite Graphs. In *IEEE Symp. on Found. of Computer Science*. pp 453–462, 1995.
- [17] N. E. Kadhi and H. El-Gendy. Advanced Method for Cryptographic Protocol Verification. *Journal of Computational Methods in Science and Engineering*, 6:109–119, 2006.
- [18] T. Legall, B. Jeannet, and T. Jhon. Verification of Communication Protocols Using Abstract Interpretation of FIFO Queues. In *Proc. of Alg. Meth. and Soft. Tech., 11th Int'l Conf. (AMAST'06) V. 4019*, LNCS, pages 263–274, 2006.
- [19] M. Leuschel. Homeomorphic Embedding for Online Termination of Symbolic Methods. In *The Essence of Computation*, volume 2566 of *LNCS*, pages 379–403, 2002.
- [20] S. Lucas. Rewriting-Based Navigation of Web Sites: Looking for Models and Logics. In *Proc. of 1st Int'l Workshop on Automated Specification and Verification of Web Sites (WWV'05)*, volume 157(2). ENTCS, Elsevier, 2005.
- [21] B. Michael, F. Juliana, and G. Patrice. Veriweb: automatically testing dynamic web sites. In *Proc. of 11th Int'l WWW Conference*. ENTCS, Elsevier, 2002.
- [22] N. Polyzotis, M. Garofalakis, and Y. E. Ioannidis. Approximate XML Query Answers. In *Proc. of the ACM Int'l Conf. on Management of Data (ICMD'04)*. pp 263–274, 2004.
- [23] TeReSe, editor. *Term Rewriting Systems*. Cambridge University Press, Cambridge, UK, 2003.