# A Semi-Automatic Methodology for Repairing Faulty Web Sites

M. Alpuente[1], D. Ballis[2], M. Falaschi[3] and J. García-Vivó[1]

[1] DSIC, Universidad Politécnica de Valencia, Camino de Vera s/n, Apdo. 22012,
46071 Valencia, Spain. Email: {alpuente, jgarciavivo}@dsic.upv.es.

[2] Dip. Matematica e Informatica, Via delle Scienze 206,
33100 Udine, Italy. Email: demis@dimi.uniud.it.

[3] Dip. de Scienze Matematiche e Informatiche. Pian dei Mantellini 44.
53100 Siena, Italy. Email: moreno.falaschi@unisi.it

# Talk Plan

- Formal Verification of Web sites
- Error Detection
- Repairing Faulty Web sites

# Talk Plan

- Formal Verification of Web sites
- Error Detection
- Repairing Faulty Web sites

# Motivation

- Web Sites can have a very complex structure
- Development and maintenance of Web sites are difficult tasks
- We use formal methods
  - to verify Web sites w.r.t a given specification, which is able to express sintactic and semantic properties
  - to fix Web sites semi-automatically

# Verification of Web sites

- On a previous work, we provided a rule-based specification language for specifying integrity conditions for a given Web site
- And a verification technique for automatically checking wether those conditions are fulfilled
- Our verification framework is based on a rewriting-like technique called partial rewriting, more suitable for dealing with XML/XHTML data

# Web site denotation

- A Web page is a ground term. Consequently, we represent a Web Site as a finite collection of ground terms of a suitable term algebra

```
<member>
    <name> Peter </name>
    <surname> Hawkins </surname>
    <status> Professor </status>
    <teaching>
        <course> Algebra </course>
    </teaching>
</member>
```

```
member(
    name("Peter")
    surname ("Hawkins")
    status ("Professor")
    teaching(
        course ("Algebra")
    )
)
```

# Web Specification

- A Web specification is made up of
  - a set of correctNess rules $I_N$
  - a set of coMpletenes rules $I_M$
  - a set of rewrite rules (i.e. a Term Rewriting System) $R$

# Correctness Rules

- A correctness rule has the following form:

  `l → error | C`

  where `l` is a term, `error` is a reserved constant and `C` is a sequence of equations and membership tests w.r.t. regular languages

  Interpretation : Given a Web site `W`, if `l` is recognized in some Web page of `W` and all the expressions represented in `C` are evaluated to True (or `C` is empty), the Web page is incorrect

e.g. project(year(X)) → error | X in [0-9]*, X<1990

# Completeness Rules

- A completeness rule has the following form:

$$l \rightarrow \mu(r)<q>$$

where l and r are terms, $\mu$ is a marking function for marking some symbols of r by means of the symbol #, and q is a universal/existential quantifier (A, E)

Marks are used to select the Web pages on which we want to check a given condition.

e.g hpage(status("Professor")) → #hpage(#status(#"Professor"),teaching)<A>

# Completeness Rules – Interpretation

- Given a Web site `W`
  - An existential completeness rule `l` $\rightarrow$ $\mu$`(r)<E>` is interpreted as follows:
    - if `l` is recognized in some Web page of `W`, then (the irreducible form of) `r` must be recognized in some Web page of `W` which contain the marked part of `r`.
  - An universal completeness rule `l` $\rightarrow$ $\mu$`(r)<A>` is interpreted as follows:
    - if `l` is recognized in some Web page of `W`, then (the irreducible form of) `r` must be recognized in every Web page of `W` which contain the marked part of `r`.

# Tree Simulation

- Simulation allows us to recognize the structure and the labels of a Web page (template) into another. It provides a powerful pattern-matching mechanism:
  - suitable for dealing with HTML/XML data (partial matching, unordered trees)
  - fast (efficient algorithms do exist)
- Minimal, injective simulations

# Partial Rewriting

- A rewriting relation in which:
  - the traditional pattern matching mechanism is replaced by tree simulation
  - the context of selected reducible expressions is disregarded
  - we deal with marking information

# Partial Rewriting steps

```
members(
    member(name(Peter),
           surname(Parker),
           status(Professor)),
    member(name(John),
           surname(Smith),
           status(technician))
)
```

is partially rewritten to

$\texttt{\#hpage(fullname(append(Peter,Parker),status)} \rightarrow_R \texttt{\#hpage(fullname(PeterParker),status)}$

and

$\texttt{\#hpage(fullname(append(John,Smith),status)} \rightarrow_R \texttt{hpage(fullname(JohnSmith),status)}$

by rule member(name(X),surname(Y))→#hpage(fullname(append(X, Y)), status)

# Talk Plan

- Formal Verification of Web sites
- Error Detection
- Repairing Faulty Web sites

# Error Detection

- Our formal verification methodology is able to detect forbidden/erroneous as well as incomplete information in a Web site `W`, by executing a Web specification on `W`.
- Kind of errors:
  - Correctness errors
  - Completeness errors
    - missing Web pages
    - Universal completeness errors
    - Existential completeness errors

# Correctness errors

- Let $\mathtt{W}$ be a Web site and $(\mathtt{I}_{M}, \mathtt{I}_{N}, \mathtt{R})$ be a Web specification. Then the triple $(\mathtt{p}, \mathtt{v}, \mathtt{l}\sigma)$ is a correctness error iff
  - $\mathtt{p} \equiv (\mathtt{V}, \mathtt{E}, \mathtt{r}, \mathtt{label}) \in \mathtt{W}$ is a Web page of $\mathtt{W}$ and $\mathtt{v} \in \mathtt{V}$ is a vertex of $\mathtt{p}$;
  - $\mathtt{l}\sigma$ is an instance of a left-hand side of a correctness rule belonging to $\mathtt{I}_{N}$ which is "embedded" in $\mathtt{p}_{|\mathtt{v}}$.
- We denote the set of all the correctness errors of a Web site risen by a set of correctness rules $\mathtt{I}_{N}$ as $\mathtt{E}_{N}$

# Completeness errors – Missing Web pages

- Let $W$ be a Web site and $(I_M, I_N, R)$ be a Web specification. Then the pair $(r, W)$ is a *missing Web page error* whenever r does not belong to W and there exists $p \in W$ s.t. $p \rightharpoonup^{+}_{I_M} r$.

# Completeness errors – Incomplete Web pages

- Let $W$ be a Web site and $(I_M, I_N, R)$ be a Web specification. Then the triple $(r, \{p_1, ..., p_n\}, A)$ is a *universal completeness error*, if there exists $p \in W$ s.t. $p \rightharpoonup^{+}_{I_M} r$ and $\{p_1, ..., p_n\}$ is not *universally complete* w.r.t $r$, $p_i \in W, i=1..n$.
- Let $W$ be a Web site and $(I_M, I_N, R)$ be a Web specification. Then the triple $(r, \{p_1, ..., p_n\}, E)$ is an *existential completeness error*, if there exists $p \in W$ s.t. $p \rightharpoonup^{+}_{I_M} r$ and $\{p_1, ..., p_n\}$ is not *existentially complete* w.r.t $r$, $p_i \in W, i=1..n$.

# Completeness errors – Incomplete Web pages

- Note that we locate where the completeness errors occur and where the information must be included
- We denote the set of all the correctness errors of a Web site risen by a set of completeness rules $I_M$ as $E_M$

# Talk Plan

- Formal Verification of Web sites
- Error Detection
- Repairing Faulty Web sites

# Repairing a Faulty Web site

- Given a Faulty Web site `W` and the sets of errors $E_N$ and $E_M$ found in that Web site, there exist several repair actions to choose between
  - `change(p,v,t)`
  - `add(p,t)`
  - `add(p,W)`
  - `delete(p,t)`
- The same error can be fixed executing different actions

# Repairing a Faulty Web site

- Our goal is to guarantee the completeness and correctness of the Web site after fixing all the errors found in the verification phase
- If $E_N$ is empty, the Web site is *Correct*
- If $E_M$ is empty, the Web site is *Complete*
- Our method is built up of several stages

# Fixing Correctness errors

- Given a correctness error `(p,v,lσ)`
  - If there exist $r \equiv (l \rightarrow error \mid C)$ in $I_N$ where `C` is empty, the only option will be `delete`
  - Otherwise, two options are avaliable: `delete` or `change`
  - $W' = W - \{p\} \cup \{delete(p,l\sigma)\}$
  - $W' = W - \{p\} \cup \{change(p,l\sigma,t)\}$

…what is `t`?

# Fixing Correctness errors

- The set of conditions to be held by the new values is computed collecting the conditions of all the rules that simulate (or are simulated by) `l`
  - project(grant1(x),grant2(y))→error | x < y
  - grant1(x) →error | x<50000
- We can use constraint programming techniques to provide sets of admissible values.
- This method lets us find inconsistencies between the rules of the Web specification, whenever no admisible values are found

# Fixing Completeness errors

- After having fixed the Correctness Errors, we need to update the set of Completeness Errors $E_M$
- Given a completeness error, it always can be repaired by applying two different actions: `add` the missing information or `delete` all the information that caused the error

# Fixing Completeness errors - `add`

- missing Web pages
  - `W' = add(r,W)`
- Existential incompleteness errors
  - $W' = W-\{p_i\} \cup \{add(p_i, r)\}$, with $p_i$ selected arbitrary from $\{p_1, \ldots, p_n\}$
- Universal incompleteness errors
  - $W' = W-\{p_i\} \cup \{add(p_i, r)\}, \forall p_i \in \{p1, \ldots pn\}$

# Fixing Completeness errors - `add`

- The missing information can be accurately calculated using existing tools (XMLDiff)
- The new information will be added to the Web site only if it does not cause new correctness errors.

# Fixing Completeness errors - `delete`

- $W' = \{delete(p, t_i), p \in W, t_1 \rightarrow t_2 \rightarrow \ldots \rightarrow r\}$

  We delete all the terms in the partial rewriting sequences that allow us to derive the requirement r

**member(name(X),surname(Y))→#hpage(name(X),#surname(#Y)) <E>**

**hpage(name(X),surname(Y))→#pubs(pub(author(Y))) <E>**

member(name(“Demis”),surname(“Ballis”))

# Fixing Completeness errors - `delete`

- $W' = \{delete(p,t_i), p \in W, t_1 \rightarrow t_2 \rightarrow \ldots \rightarrow r\}$

  We delete all the terms in the partial rewriting sequences that allow us to derive the requirement r

**member(name(X),surname(Y))→#hpage(name(X),#surname(#Y)) <E>**

**hpage(name(X),surname(Y))→#pubs(pub(author(Y))) <E>**

member(name("Demis"),surname("Ballis")) →hpage(name("Demis"),surname("Ballis"))

# Fixing Completeness errors - `delete`

- $W' = \{delete(p,t_i) , p \in W, t_1 \rightarrow t_2 \rightarrow \ldots \rightarrow r\}$

  We delete all the terms in the partial rewriting sequences that allow us to derive the requirement r

**member(name(X),surname(Y))→#hpage(name(X),#surname(#Y)) <E>**

**hpage(name(X),surname(Y))→#pubs(pub(author(Y))) <E>**

member(name("Demis"),surname("Ballis")) →hpage(name("Demis"),surname("Ballis"))

hpage(name("Demis"),surname("Ballis"))

# Fixing Completeness errors - `delete`

- $W' = \{delete(p,t_i), p \in W, t_1 \rightarrow t_2 \rightarrow \ldots \rightarrow r\}$

  We delete all the terms in the partial rewriting sequences that allow us to derive the requirement r

**member(name(X),surname(Y))→#hpage(name(X),#surname(#Y)) <E>**

**hpage(name(X),surname(Y))→#pubs(pub(author(Y))) <E>**

member(name("Demis"),surname("Ballis")) →hpage(name("Demis"),surname("Ballis"))

hpage(name("Demis"),surname("Ballis")) → **pubs(pub(author("Ballis")))**

# Fixing Completeness errors - `delete`

- $W' = \{delete(p,t_i), p \in W, t_1 \rightharpoonup t_2 \rightharpoonup \ldots \rightharpoonup r\}$

  We delete all the terms in the partial rewriting sequences that allow us to derive the requirement r

**member(name(X),surname(Y))→#hpage(name(X),#surname(#Y)) <E>**

**hpage(name(X),surname(Y))→#pubs(pub(author(Y))) <E>**

member(name(“Demis”),surname(“Ballis”)) →hpage(name(“Demis”),surname(“Ballis”))

~~hpage(name(“Demis”),surname(“Ballis”))~~ → **pubs(pub(author(“Ballis”)**

# Fixing Completeness errors - `delete`

- $W' = \{delete(p, t_i), p \in W, t_1 \rightharpoonup t_2 \rightharpoonup \ldots \rightharpoonup r\}$

  We delete all the terms in the partial rewriting sequences that allow us to derive the requirement r

**member(name(X),surname(Y))→#hpage(name(X),#surname(#Y)) <E>**

**hpage(name(X),surname(Y))→#pubs(pub(author(Y))) <E>**

member(name("Demis"),surname("Ballis")) →hpage(name("Demis"),surname("Ballis"))

~~hpage(name("Demis"),surname("Ballis"))~~ → pubs(pub(author("Ballis"))

# Fixing Completeness errors - `delete`

- $W' = \{delete(p, t_i), p \in W, t_1 \rightharpoonup t_2 \rightharpoonup \ldots \rightharpoonup r\}$

  We delete all the terms in the partial rewriting sequences that allow us to derive the requirement r

**member(name(X),surname(Y))→#hpage(name(X),#surname(#Y)) <E>**

**hpage(name(X),surname(Y))→#pubs(pub(author(Y))) <E>**

~~member(name("Demis"),surname("Ballis"))~~ → hpage(name("Demis"),surname("Ballis"))

~~hpage(name("Demis"),surname("Ballis"))~~ → pubs(pub(author("Ballis"))

# Fixing Completeness errors - `delete`

- If the completeness rule that found the error has no variables, this method will probably delete too much information

  e.g. hpage() → hpage(title(body()))<A>

# Conclusions

- On a previous work, we presented a rule-based verification methodology
- Now, we have presented a novel methodology to repair a faulty Web site w.r.t a Web specification
- After applying this method, we guarantee that the Web site is Correct and Complete

# Future Work

- Implementation of the repairing methodology
- Implementation of an intuitive, graphical language for the definition of the Web site specifications (e.g. correctness, completeness properties)
- Implementation of an adaptive, intelligent tutor for both tasks (defining properties and reparing the Web site)

# END