Semantic Verification of Web System Contents

M. Alpuente[1] M. Baggi[2] D. Ballis[3] M. Falaschi[2]

[1] DSIC, Universidad Politecnica de Valencia, Spain.
alpuente@dsic.upv.es
[2] University of Siena, Italy.
{baggi,moreno.falaschi}@unisi.it
[3] DIMI, University of Udine, Italy.
demis@dimi.uniud.it

Fifth International Workshop on Web Information Systems Modeling (WISM '08)

Outline

Proposal

Goal

We provide a rule-based specification language to formalize and automatically check syntactic and semantic properties over the static contents of any Web system.

- combining ontology reasoning with correctness and completeness rule specification;
- use semantic information to model semantic properties;
- a novel verification methodology to check a specification against the considered Web contents.

Motivation

Web systems are very often collaborative applications in which many users freely contribute to update their contents (e.g. wikis, blogs, social networks, etc.). In this scenario,

- keeping data correct and complete is arduous;
- there is a very poor control over the content update operation;
- data inconsistency problems easily arise.

To facilitate the task of data control we need automatic tools to specify properties over data contents and check them.

Motivation

Web systems are very often collaborative applications in which many users freely contribute to update their contents (e.g. wikis, blogs, social networks, etc.). In this scenario,

- keeping data correct and complete is arduous;
- there is a very poor control over the content update operation;
- data inconsistency problems easily arise.

To facilitate the task of data control we need automatic tools to specify properties over data contents and check them.

Most of the approaches to solve this problem only rely on the data to be checked.

Our Contribution

We provide a language inspired by the GVerdi specification language, that we extend in the following ways.

- new rule constructs to improve the language expressiveness;
- queries to multiple ontologies to retrieve semantic information;
- integration of semantic information in specification rules to
  - express semantic conditions,
  - use meta-symbols in rule specification;
- novel verification methodology to check specifications.

DIG

The connection with the ontology reasoner is obtained by means of the DIG interface, which is a standard API for description logic systems.

DIG Languages

The DIG interface is equipped with four XML languages employed to formalize and query ontologies.

- Concept language used to build roles, individuals and complex concepts
- Tell language used to describe ontologies to reasoners
- Ask language used to query ontologies
- Response language used to formalize query responses

Extended DIG Ask Statements

We adopted a generalized version of DIG ask statements by defining "templates"

Extensions

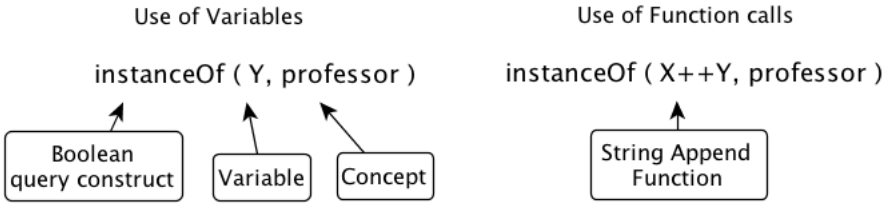
- Variables employed as placeholders for atomic concepts, roles and individuals
- Function calls compute atomic concepts, roles and individuals.

Extended DIG Ask Statements

We adopted a generalized version of DIG ask statements by defining "templates"

Extensions

- Variables employed as placeholders for atomic concepts, roles and individuals
- Function calls compute atomic concepts, roles and individuals.



Specification Language

The language provides constructs for specifying two kinds of rules: correctness and completeness rules. Both kinds may be conditionals, that is, they can be fired iff the associated condition holds.

Condition

A condition is a finite sequence $c_1, \ldots, c_n$, where each c_i can be:

- membership test w.r.t. regular expression (e.g. $\mathtt{X} \in \mathtt{rexp}$)
- equation $s = t$, where s, t are expressions which may contain nested function calls
- a boolean ontology query

Correctness Rule

A *correctness rule* is an expression of the form

$$\bigwedge_{i=1}^{n} \mathtt{l}_i \rightharpoonup \mathtt{error} \mid \mathtt{C}$$

where each $\mathtt{l}_i$ is a term, $\mathtt{error}$ is a reserved constant, $\mathtt{C}$ is a condition.

Meaning

Whenever an instance $\mathtt{l}_i\sigma$ of $\mathtt{l}_i$ for each $i \in \{1, \ldots, n\}$ is recognized in some Web page $\mathtt{p}$, and the rule condition $\mathtt{C}$ holds for σ, then Web page $\mathtt{p}$ is signaled as an incorrect page.

Example

Property to be specified

If an associate professor has more than three Ph.D. students, then he cannot teach more than one course.

Formalization

```
use './Ontologies/UniversityDomain' as univ
course(cId(X),prof(name(Y)))∧course(cId(Z),prof(name(Y)))⇀error
  | univ.instanceOf(Y,AssocProf⊓(≥3hasStd⊓∀hasStd.PhDStudent)),
   X =/= Z
```

Completeness Rule

A *completeness rule* is an expression of the form

$$\bigwedge_{i=1}^{n} \mathtt{l}_i \rightharpoonup \bigvee_{j=1}^{m} \mathtt{r}_j \,|\, \mathtt{C}\ \mathtt{containing}\ \mathtt{ct}\ \langle \mathtt{q} \rangle$$

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

Meaning

If an instance $\mathtt{l}_i\sigma$ of $\mathtt{l}_i$ for each $i \in \{1, \dots, n\}$ is recognized in some Web page p and the condition $\mathtt{C}$ holds for σ, then an instance $\mathtt{r}_j\sigma$ of at least one $\mathtt{r}_j$, $j \in \{1, \dots, m\}$ must be recognized in *all* (resp. *some*) Web pages containing the $\mathtt{ct}$ term.

Example

Property to be specified

For each course, given by a full professor, at least two exam dates must be provided.

Formalization

```
use './Ontologies/UniversityDomain' as univ
course(cId(X)) ⇀ course(cId(X),examDate(),examDate())
       | univ.instanceOf(X,∃CourseGivenBy.FullProf) <E>
```

Met-symbols

- Completeness and correctness rules may include special meta-symbols into those terms which are associated with non-boolean ontology queries.
- Web specification rules containing meta-symbols have to be pre-processed in order to expand them to a set of rules without meta-symbols, before being executed.

Met-symbols

- Completeness and correctness rules may include special meta-symbols into those terms which are associated with non-boolean ontology queries.
- Web specification rules containing meta-symbols have to be pre-processed in order to expand them to a set of rules without meta-symbols, before being executed.

E.g.

```
metasymbol prof:  univ.getChildren("Professor")
```
expands to `{FullProf, AssociateProf }`

Example

Property to be specified

We want that email or post address have to be specified for each university professor.

Formalization

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

Example (2)

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

```
use './Ontology/UniversityDomain' as univ
AssociateProf(name(X)) ⇀ AssociateProf(name(X),email()) ∨
                          AssociateProf(name(X),address()) |
                                   containing member() <A>
FullProf(name(X)) ⇀ FullProf(name(X),email()) ∨
                     FullProf(name(X),address()) |
                                   containing member() <A>
```

Simplified Rule Semantics

Correctness rules

A -> error

Semantics: if an instance of term A is detected, then an error arises.

Completeness rules

A -> B

Semantics: if an instance of term A is detected, then, there should be also the corresponding instance of term B.

Detect Correctness Errors

Incorrectness symptom

To detect correctness errors we just need to execute the set of correctness rules over the Web system contents and whenever a rule is fired, the detected instance of the left hand side ($\bigwedge_{i=1}^{n} \mathtt{l}_i$) provides an *incorrectness symptom*

Detect Completeness Errors

Web Content:
A
Completeness Rule Set:
r_1: A -> B
r_2: B -> C

Detect Completeness Errors

Web Content:
A
Completeness Rule Set:
r_1: A -> B
r_2: B -> C

Rule Application

If the rule set is applied to the Web Content only once, we obtain the completeness error B.

Detect Completeness Errors

Let's correct the Web Content adding information B.

Web Content:
A, B
Completeness Rule Set:
r_1: A -> B
r_2: B -> C

Detect Completeness Errors

Let's correct the Web Content adding information B.

Web Content:
A, B
Completeness Rule Set:
r_1: A -> B
r_2: B -> C

Rule Application

If the rule set is applied again to the Web Content, we obtain the completeness error C.

A Structured Way to Proceed

Derivation tree

Starting from the Web content information, let’s compute the whole set of required information induced by the completeness rule set, that is, the information which makes the Web Content complete w.r.t. the specification.

A Structured Way to Proceed

Derivation tree

Starting from the Web content information, let's compute the whole set of required information induced by the completeness rule set, that is, the information which makes the Web Content complete w.r.t. the specification.

Web Content:
A
Completeness Rule Set:
r_1: A -> B
r_2: B -> C

Required information

The set {A, B, C}.

Derivation tree

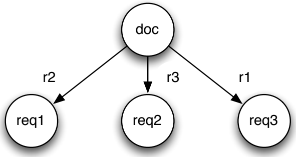
Derivation tree construction

Start from the Web content information at the root tree, for instance a document.

Derivation tree

Derivation tree construction (2)

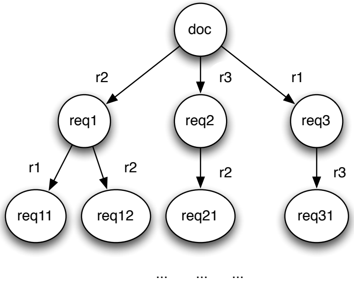
Apply the completeness rules to ’doc’ and add the obtained required information as its children.


$$\begin{tikzcd}
 & \text{doc} \arrow[dl, "r2"'] \arrow[d, "r3"] \arrow[dr, "r1"] & \\
\text{req1} & \text{req2} & \text{req3}
\end{tikzcd}$$

Derivation tree

Derivation tree construction (3)

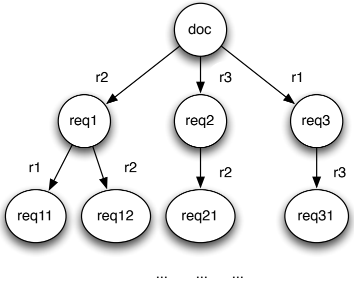
Apply the completeness rules to all the leaves and add the obtained required information as their children.



Derivation tree

Derivation tree construction (3)

Apply the completeness rules to all the leaves and add the obtained required information as their children.



Does it terminate?

Termination Assurance

Cyclic-Activated Rules

- h(X) -> h(h(X)) and information h(a)

Termination Assurance

Cyclic-Activated Rules

- h(X) -> h(h(X)) and information h(a)

  h(a)

Termination Assurance

Cyclic-Activated Rules

- h(X) -> h(h(X)) and information h(a)

  h(a) -> h(h(a))

Termination Assurance

Cyclic-Activated Rules

- h(X) -> h(h(X)) and information h(a)

  h(a) -> h(h(a)) -> h(h(h(a))) -> ...

Termination Assurance

Cyclic-Activated Rules

- h(X) -> h(h(X)) and information h(a)

  h(a) -> h(h(a)) -> h(h(h(a))) -> ...

- p(X) -> m(h(X)), h(Y) -> p(Y) and information p(a)

Termination Assurance

Cyclic-Activated Rules

- h(X) -> h(h(X)) and information h(a)

  h(a) -> h(h(a)) -> h(h(h(a))) -> ...

- p(X) -> m(h(X)), h(Y) -> p(Y) and information p(a)

  p(a)

Termination Assurance

Cyclic-Activated Rules

- h(X) -> h(h(X)) and information h(a)

  h(a) -> h(h(a)) -> h(h(h(a))) -> ...

- p(X) -> m(h(X)), h(Y) -> p(Y) and information p(a)

  p(a) -> (1) m(h(a))

Termination Assurance

Cyclic-Activated Rules

- h(X) -> h(h(X)) and information h(a)

  h(a) -> h(h(a)) -> h(h(h(a))) -> ...

- p(X) -> m(h(X)), h(Y) -> p(Y) and information p(a)

  p(a) -> (1) m(h(a))-> (2) m(p(a))

Termination Assurance

Cyclic-Activated Rules

- h(X) -> h(h(X)) and information h(a)

  h(a) -> h(h(a)) -> h(h(h(a))) -> ...

- p(X) -> m(h(X)), h(Y) -> p(Y) and information p(a)

  p(a) -> (1) m(h(a))-> (2) m(p(a))-> (1) m(m(h(a)))

Termination Assurance

Cyclic-Activated Rules

- h(X) -> h(h(X)) and information h(a)

  h(a) -> h(h(a)) -> h(h(h(a))) -> ...

- p(X) -> m(h(X)), h(Y) -> p(Y) and information p(a)

  p(a) -> (1) m(h(a))-> (2) m(p(a))-> (1) m(m(h(a)))-> (2) m(m(p(a))) -> ...

Termination Assurance

Cyclic-Activated Rules

- h(X) -> h(h(X)) and information h(a)

  h(a) -> h(h(a)) -> h(h(h(a))) -> ...

- p(X) -> m(h(X)), h(Y) -> p(Y) and information p(a)

  p(a) -> (1) m(h(a))-> (2) m(p(a))-> (1) m(m(h(a)))-> (2) m(m(p(a))) -> ...

Syntactic Restriction

Completeness rule sets which are cyclic-activated are avoided.

Conclusions & Future work

Conclusions

We provided a specification language to formalize and check syntactic and semantic properties over Web system contents, exploiting ontology reasoning to retrieve semantic information. Moreover we introduced a novel verification methodology to check the specification against considered Web contents.

Future work

An implementation of such a langugare and the related verification system is on going.