Management of Change in MAYA

Serge Autexier\textsuperscript{1,2}, Dieter Hutter\textsuperscript{2}, Till Mossakowski\textsuperscript{3}, Axel Schairer\textsuperscript{2}

\textsuperscript{1} Saarland University, Saarbrücken, Germany
\textsuperscript{2} German Research Center for Artificial Intelligence, Saarbrücken, Germany
\textsuperscript{3} University of Bremen, Germany

Mathematics on the Semantic Web
Eindhoven, The Netherlands
Evolutionary Formal Software Development

- **Changing specifications due to proof failures**
- **Translation into logical representation**
- **Internal Representation**
- **Theorem Prover**
- **Proof Obligations + Structured Database**

Source: Serge Autexier
spec LIST [sort Elem] =
    free type List[Elem] ::= [] | ___ :: ___(Elem; List[Elem])
ops __ + +___ : List[Elem] × List[Elem] → List[Elem];
    reverse : List[Elem] → List[Elem];
pred null : List[Elem]
∀ x, y : Elem;
    K, L : List[Elem]
. [] + +K = K
. (x :: L) + +K = x :: (L + +K)
. reverse([]) = []
. reverse(x :: L) = reverse(L) + + (x :: [])
. null(L) ↔ L = []

then %implies
∀ K, L : List[Elem]. reverse(K + +L) = reverse(L) + + reverse(K)
    . null(reverse(L)) ↔ null(L)
end

spec MONOID =
    sort Elem
ops e : Elem;
    ___ * ___ : Elem × Elem → Elem, assoc, unit e
end

view MONOID AS LIST : MONOID to LIST [sort Elem] =
    Elem ↔ List[Elem],
    e ↔ [],
    ___ * ___ ↔ ___ + +___
end
Evolutionary Formal Software Development

- Structured Specification
  - Changing specifications due to proof failures
  - Translation into logical representation

- Proof Obligations + Structured Database

- Theorem Prover
- Internal Representation
LIST[Elem]
Local Signature:

- Local sorts List[Elem]
- Local constants [] : List[Elem], ...

Local Axioms: [] ++ K = K, ...
Local Lemmata:
- reverse(K ++ L) = reverse(L) ++ reverse(K)
- ...

MONOID
Local Signature:

- Local sorts: Elem
- Local constants: * : Elem × Elem → Elem

Local Axioms: x * e = x, ...

Development Graphs introduced by [Hutter 2000]

Source: Serge Autexier
Global links from $N$ to $M$ import complete signature and axioms from $N$

Local links import local signature and axioms only

Used to represent instantiation of parameterized specifications

```
LIST
list, nil, cons, ...
```

```
ELEM
elem, < ...
```
Development Graphs II

- Global links from $N$ to $M$ import complete signature and axioms from $N$
- Local links import local signature and axioms only
  Used to represent instantiation of parameterized specifications

**Diagram:**

- **LIST**
  - $(\text{list}, \langle \text{nil, cons, \ldots} \rangle)$

- **ELEM**
  - $(\text{elem}, \langle < \ldots \rangle)$

- **NAT**
  - $(\text{nat}, \langle 0, \text{succ} \ldots \rangle)$
Global links from $N$ to $M$ import complete signature and axioms from $N$

Local links import local signature and axioms only

Used to represent instantiation of parameterized specifications
Development Graphs II

- Global links from $N$ to $M$ import complete signature and axioms from $N$
- Local links import local signature and axioms only

Used to represent instantiation of parameterized specifications

- **LIST**
  - $\langle \text{list}, \langle \text{nil, cons}, \ldots \rangle \rangle$

- **NATLIST**
  - $\langle \text{Nlist}, \langle \text{Nnil, Ncons}, \ldots \rangle \rangle$

- **ELEM**
  - $\langle \text{elem}, \langle < \ldots \rangle \rangle$

- **NAT**
  - $\langle \text{nat}, \langle 0, \text{succ} \ldots \rangle \rangle$
Global links from $N$ to $M$ import complete signature and axioms from $N$

Local links import local signature and axioms only

Used to represent instantiation of parameterized specifications
Global links from $N$ to $M$ import complete signature and axioms from $N$

Local links import local signature and axioms only

Used to represent instantiation of parameterized specifications
Development Graphs III

Development Graph

= Structured Logical Content of Specifications
  + Status of proof obligations
    (pending, proven, used axioms, ...)

Structured Specification
Verification +
Management
Evolutionary Formal Software Development

Structured Specification

Theorem Prover

Structured Internal Representation

Proof Obligations + Structured Database

Changing specifications due to proof failures

Translation into logical representation
Exploiting the Graph Structure

**STACK**  
<stack> , <empty, ...>

**LIST**  
<list> , <nil, cons>

**NATLIST**  
<Nlist> , <Nnil, Ncons>

**NATSTACK**  
<Nstack> , <Nempty, ...>

**ELEM**  
<elem> , <>

**NAT**  
<nat> , <0, succ>

Source: Serge Autexier

Mathematics on the Semantic Web – p.14
Exploiting the Graph Structure

Proof obligations:

- all axioms defining STACK in LIST
- all axioms defining STACK in NATSTACK
- all axioms defining ELEM in NAT
- all axioms defining NATSTACK in NATLIST
- all axioms defining LIST in NATLIST
Decomposition of Global Links into Local Links
Exploiting the Graph Structure

- STACK: <stack>, <empty, ...>
- LIST: <list>, <nil, cons>
- NATLIST: <Nlist>, <Nnil, Ncons>
- NATSTACK: <Nstack>, <Nempty, ...>
- ELEM: <elem>, <>
- NAT: <nat>, <0, succ>

stack \rightarrow Nstack, ...
list \rightarrow Nlist, ...
Nstack \rightarrow Nlist, ...
elem \rightarrow nat
Exploiting the Graph Structure

Source: Serge Autexier
Exploiting the Graph Structure

Subsumption of Links by Paths
Exploiting the Graph Structure

- STACK: \(<\text{stack}>, \text{empty}, \ldots>\)
- LIST: \(<\text{list}>, \text{nil}, \text{cons}>\)
- NATLIST: \(<\text{Nlist}>, \text{Nnil}, \text{Ncons}>\)
- NATSTACK: \(<\text{Nstack}>, \text{Nempty}, \ldots>\)
- ELEM: \(<\text{elem}>, \text{<>}>\)
- NAT: \(<\text{nat}>, \text{<0, succ}>\)

Mathematics on the Semantic Web – p.20

Source: Serge Autexier
Exploiting the Graph Structure

8 Proof obligations:

- Local axioms from STACK in LIST
- Local axioms from ELEM in NAT

Reduction of $\approx 75\%$ by exploiting graph structure
Structured Verification

- Exploiting the structure reduces amount of proof obligations drastically
- Indispensable to deal with effects of correcting flaws
- Remaining proof obligations must be tackled by some theorem prover
Structured Verification

Verification in-the-large

- Exploiting the structure reduces amount of proof obligations drastically
- Indispensable to deal with effects of correcting flaws

- Remaining proof obligations must be tackled by some theorem prover
Structured Verification

Verifikation in-the-large

- Exploiting the structure reduces amount of proof obligations drastically
- Indispensable to deal with effects of correcting flaws

Verifikation in-the-small

- Remaining proof obligations must be tackled by some theorem prover
State of the Art theorem provers deal
  - with verification in-the-small but
  - not or only to some small extend with verification in-the-large

Need for theorem prover for verification in-the-large

MAYA has been designed to be an add-on to theorem provers with full support for verification in-the-large
Unwinding Evolutionary Formal Software Development

Development Graph = Structured Representation + Decomposition + Proofs of theorems
More Closely... 

**SPEC** $i$

Structured Rep. $i$

Development Graph $i$

**SPEC** $i + 1$

Structured Rep. $i + 1$

Development Graph $i + 1$

$\Delta$ : Basic operations to adapt development graph

- Difference Analysis to compute basic operations

- Execution of basic steps followed by strategies guiding verification-in-the large
  - to preserve proofs, link decompositions & link subsumptions
  - to derive new link decomposition & new link subsumptions
Distributed Formal Software Developments

- **SPEC 1** → Structured Rep. 1 → Development Graph 1
- **SPEC** → Structured Rep. → Development Graph 1’
- **SPEC 2** → Structured Rep. 2 → Development Graph 2
- **SPEC** (Source: Serge Autexier)
1. Maintains structured formal developments
   - uniform and structured representation
   - explicit representation of axiomatic and postulated relationships

2. Difference Analysis between Structured Developments
   - analyses differences between old and new translated specification
   - computes set of basic operations that are necessary to adapt development graph

3. Theorem prover for verification in-the-large
   - Calculus to reason about the graph structure
   - strategies for decomposition & subsumption of links
   - strategies to preserve information about link decompositions, link subsumption and in-the-small proofs of theorems after changes
MAYA’s Social Life

Parsers

- VSE-SL/TL
- CASL
- OMDOC
- Your Parser

Theorem Provers, Decision Procedures, ...

- CORE
- INKA
- OMEGA
- HOL-CASL
- MONA
- Your System

Databases

- MBASE
Talking to MAYA...

- **Interface for Parsers for language S:**
  - Define translation from S into development graphs
  - Prove adequacy (or at least soundness) of translation
  - Implement translation

- **Interface for Systems (prover, etc.) with logic $\mathcal{L}$:**
  - Logic morphism from MAYA’s logic (currently HOL) to $\mathcal{L}$
  - $\rightarrow$: Insertion/Deletion of signature declarations, axioms, prove conjecture
  - $\leftarrow$: Proof Information: Proved?, axioms used in proof.
  - Typical Problem: non-monotonic update of theorem prover DB
    $\Rightarrow$ Several possible integration scenarios
Related Work

- Development graph in KIV:
  - Tailored to KIV specification languages
  - More adequate representation of proof obligations
  - Hampers use of different specification language
  - Lacks mechanism for decomposition and subsumption

- SPECWARE system
  - Tailored to SPECWARE specification language
  - Lacks mechanism for decomposition and subsumption, and even maintenance of established proof obligations.

- Little Theories
  - Similar global structuring links
  - More general so far as it supports heterogenous graphs
  - Lacks ability to represent intermediate states
  - Lacks mechanisms for decomposition and subsumption, no management of change
Conclusion Structured Developments

- Exploiting the structure of specifications is essential to reduce proof obligations
- Essential to deal with effects of changes in specifications
- Both can be automatically supported by theorem prover on the structured representation (verification in-the-large)
Summary MAYA

- “Truth-Maintenance System” for structured developments
- Propagation of textual changes to changes in logical representation
- Propagation of changes to the validity of proofs
  - Dependency analysis + Timestamps
- Uniform interface to theorem provers
Ongoing and Future Work

- Support hiding, heterogenous development graphs
  [FASE’01, FOSSACS’02]
- Generate explicit proof-objects for whole developments
  (independent proof checking)
- Maintaining domain specific tactical knowledge of theorem provers
- Integrate further specialized provers / decision procedures
- Lemmas speculation exploiting graph structure
- Reuse of proofs