Management of Change in MAYA

Serge Autexier[1,2], Dieter Hutter[2], Till Mossakowski[3], Axel Schairer[2]

[1] Saarland University, Saarbrücken, Germany

[2] German Research Center for Artificial Intelligence, Saarbrücken, Germany

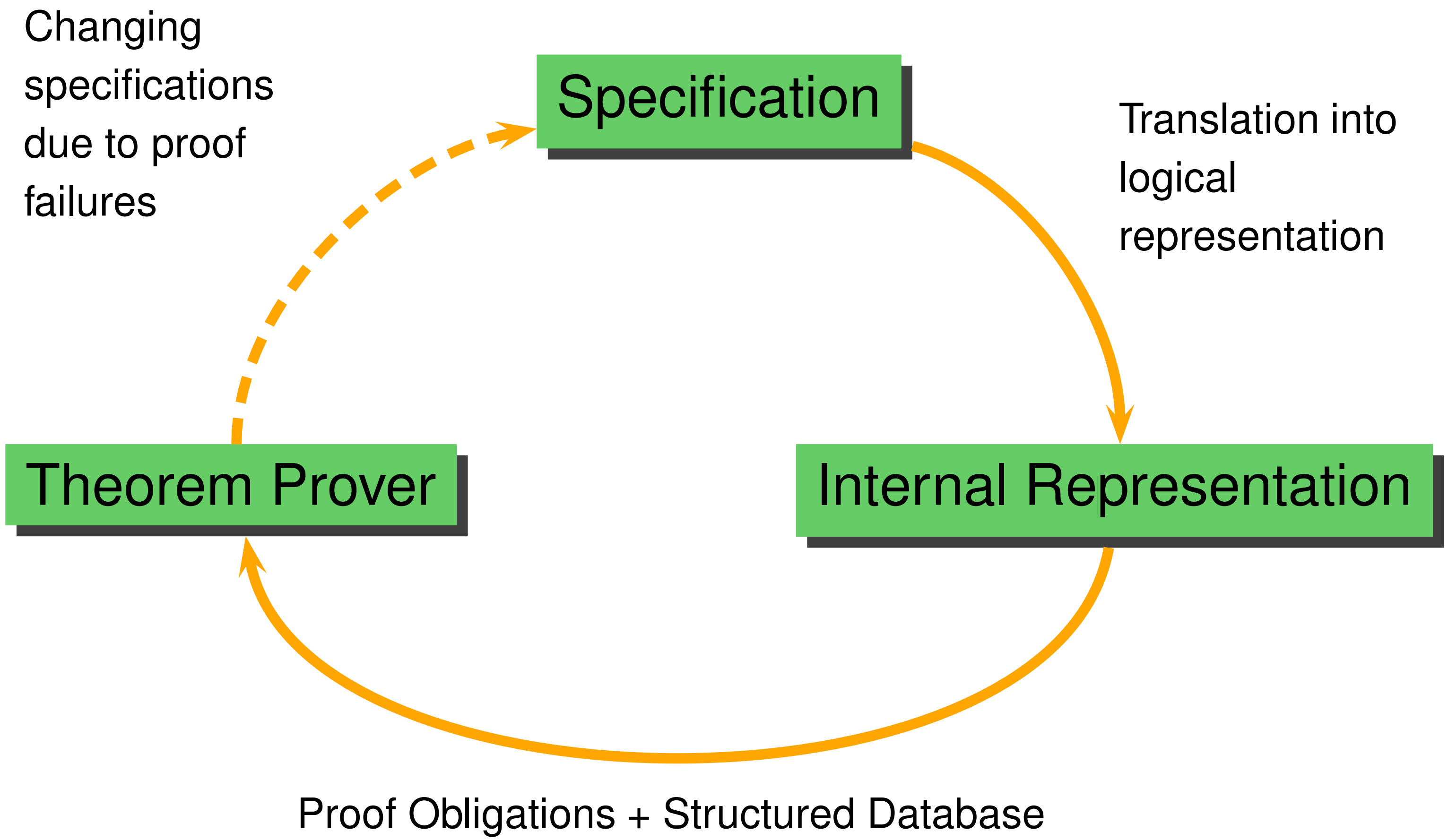
[3] University of Bremen, Germany

Mathematics on the Semantic Web

Eindhoven, The Netherlands

Context

Evolutionary Formal Software Development



Source: Serge Autexier

CASL Specifications

MAYA

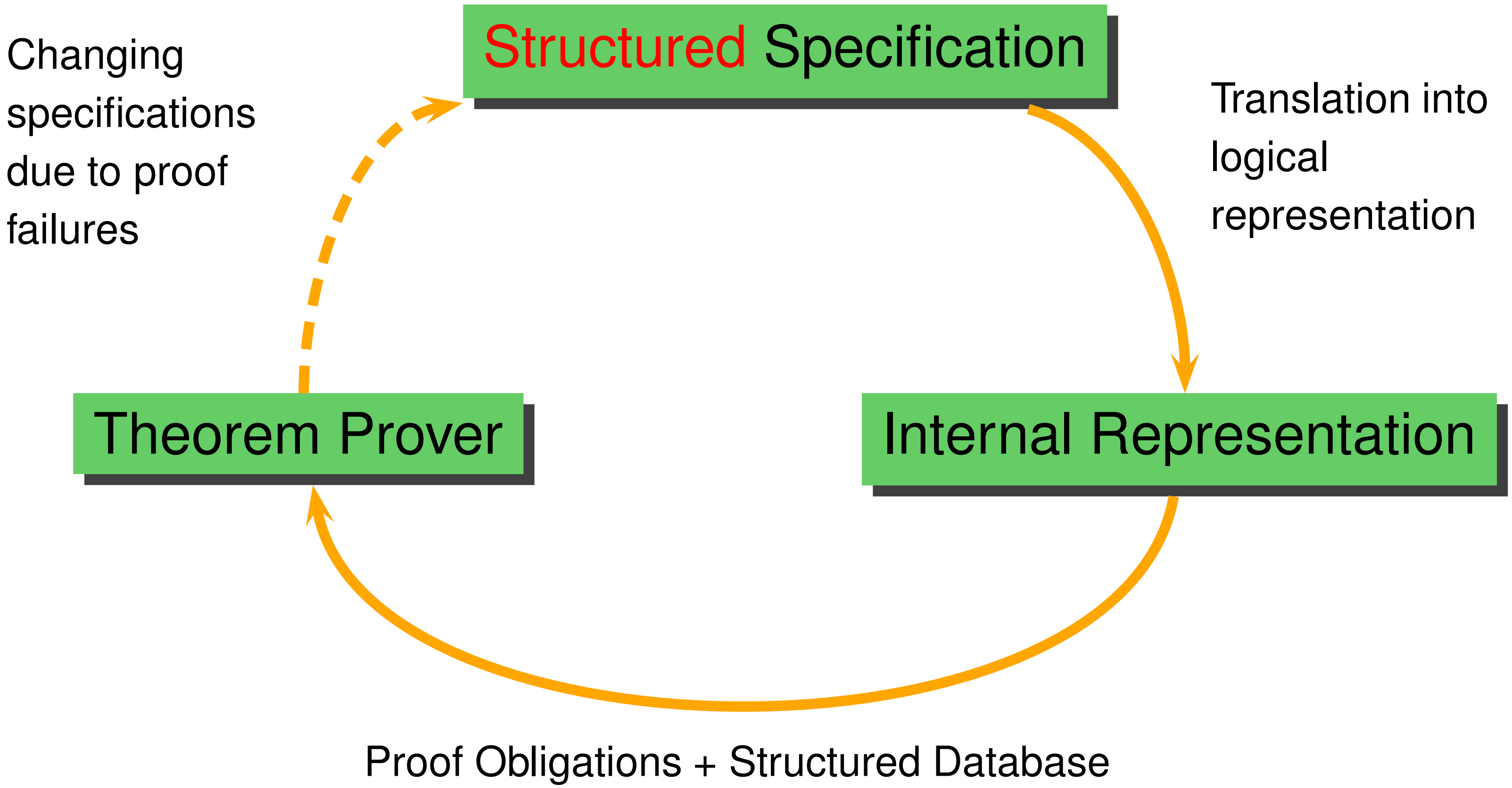
spec LIST [**sort** Elem] =
  free type List[Elem] ::= [] | __ :: __(Elem; List[Elem])
  ops $\_\_ + + \_\_$: List[Elem] $\times$ List[Elem] $\rightarrow$ List[Elem];
    reverse : List[Elem] $\rightarrow$ List[Elem];
  pred null : List[Elem]
  $\forall$ x, y : Elem;
    K, L : List[Elem]
  . [] + +K = K %(concat_nil_List)%
  . (x :: L) + +K = x :: (L + +K) %(concat_NeList_List)%
  . reverse([]) = [] %(reverse_nil)%
  . reverse(x :: L) = reverse(L) + +(x :: []) %(reverse_NeList)%
  . null(L) $\Leftrightarrow$ L = [] %(null)%
then %implies
  $\forall$ K,L : List[Elem] . reverse(K + +L) = reverse(L) + +reverse(K)
    . null(reverse(L)) $\Leftrightarrow$ null(L)
end

spec MONOID =
  sort Elem
  ops e : Elem;
    $\_\_ * \_\_$: Elem $\times$ Elem $\rightarrow$ Elem, **assoc**, **unit** e
end

view MONOIDASLIST : MONOID **to** LIST[**sort** Elem] =
  Elem $\mapsto$ List[Elem],
  e $\mapsto$ [],
  __*__ $\mapsto$ __++__
end

Context

Evolutionary Formal Software Development



Source: Serge Autexier

Development Graphs I

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)
- . . .

σ

MONOID

Local Signature:

- Local sorts: Elem
- Local constants: $*$: Elem $\times$ Elem $\rightarrow$ Elem

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

id

Local Signature:

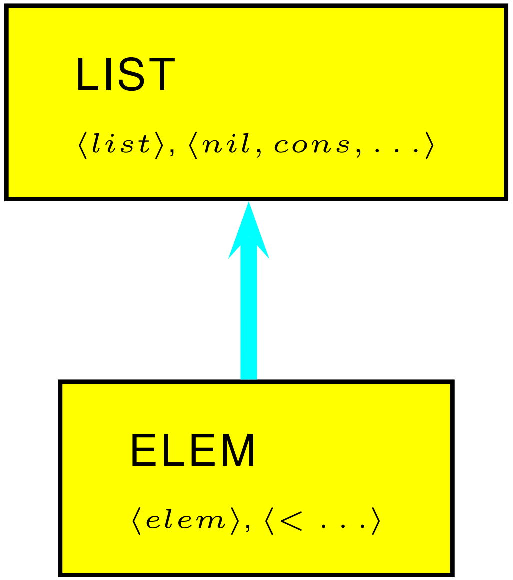
- Local sorts: Elem

Development Graphs introduced by [Hutter 2000]

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

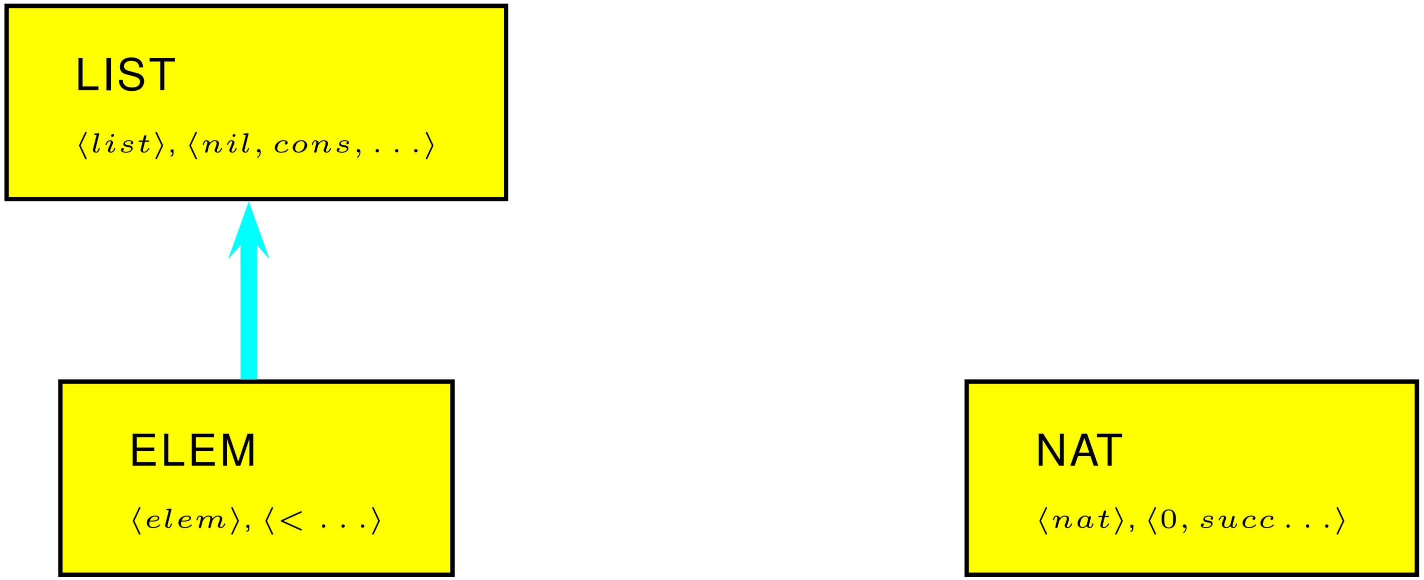


Source: Serge Autexier

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



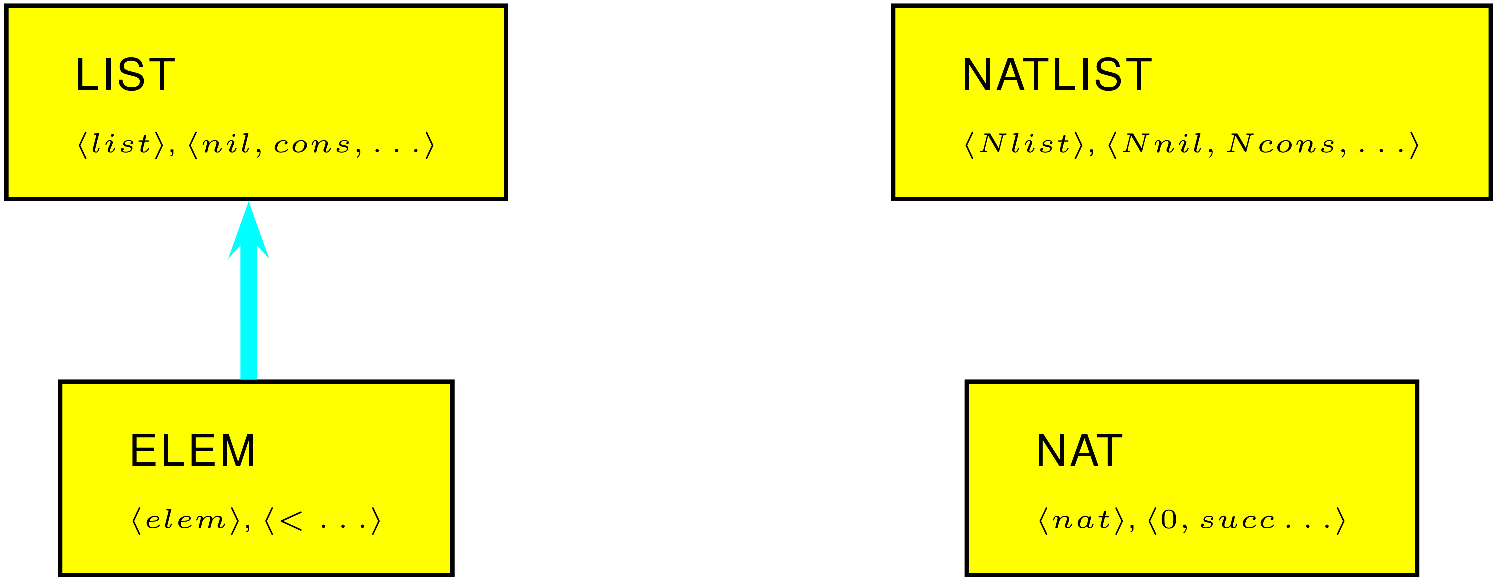
Source: Serge Autexier

Development Graphs II

MAYA

- 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



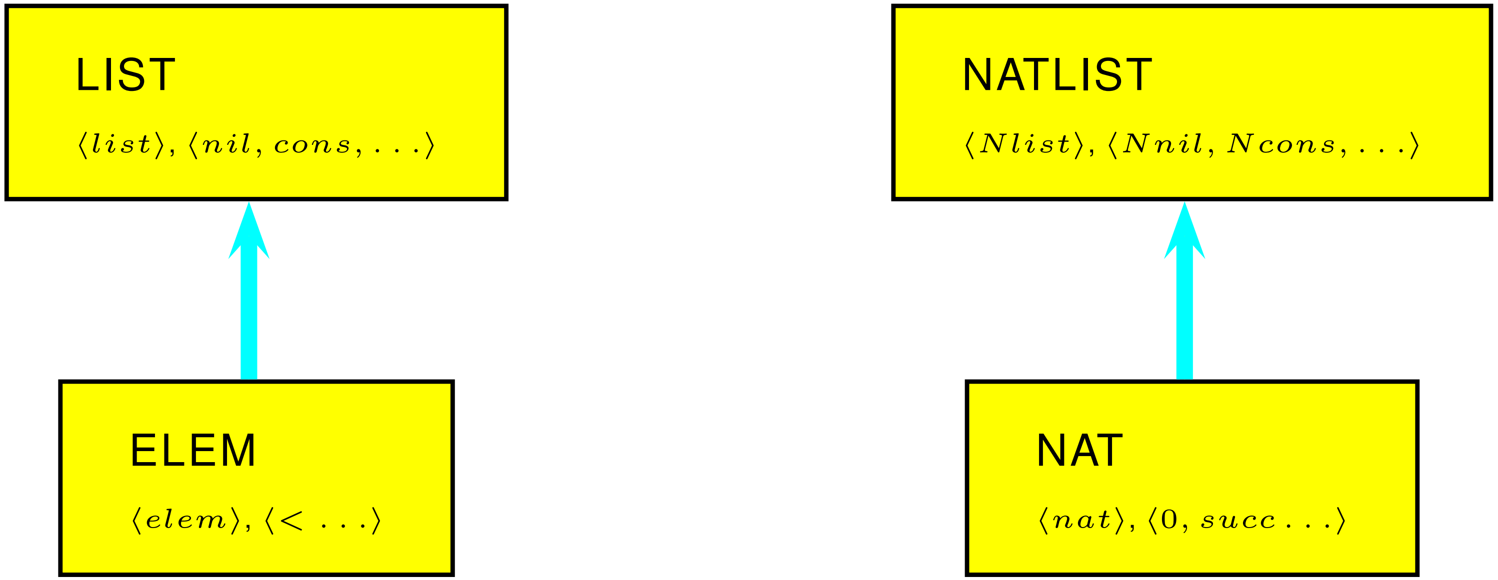
Source: Serge Autexier

Development Graphs II

MAYA

- 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

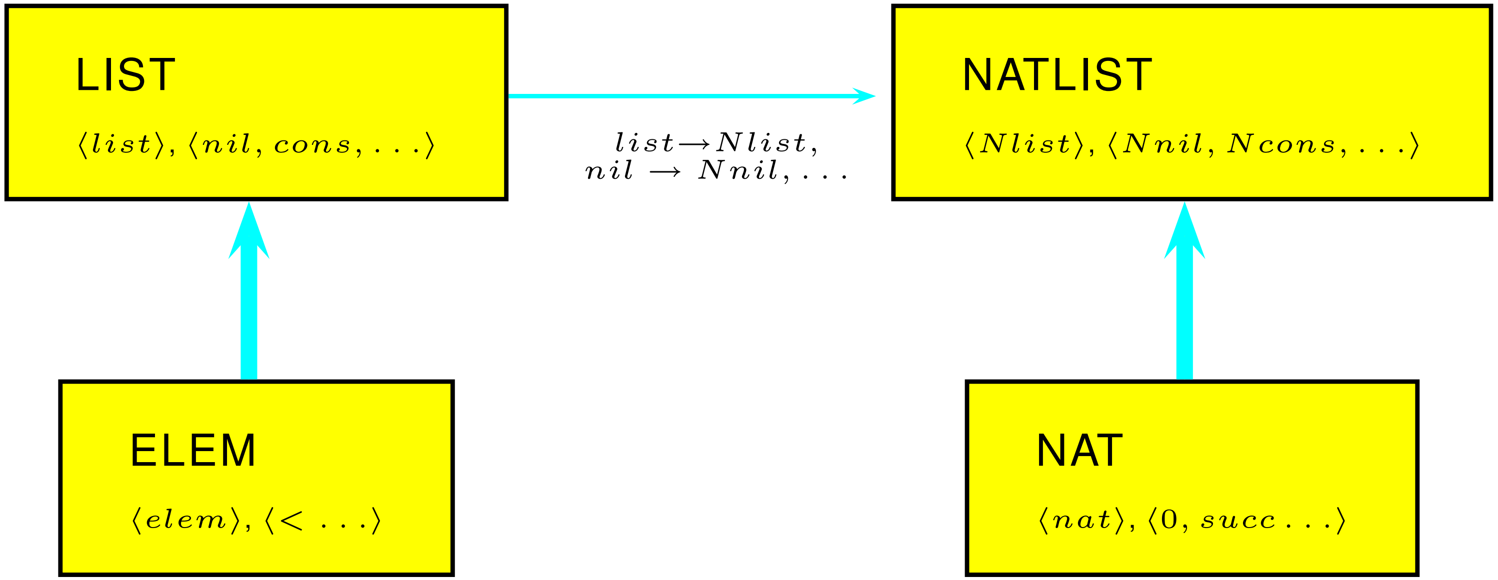


Source: Serge Autexier

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

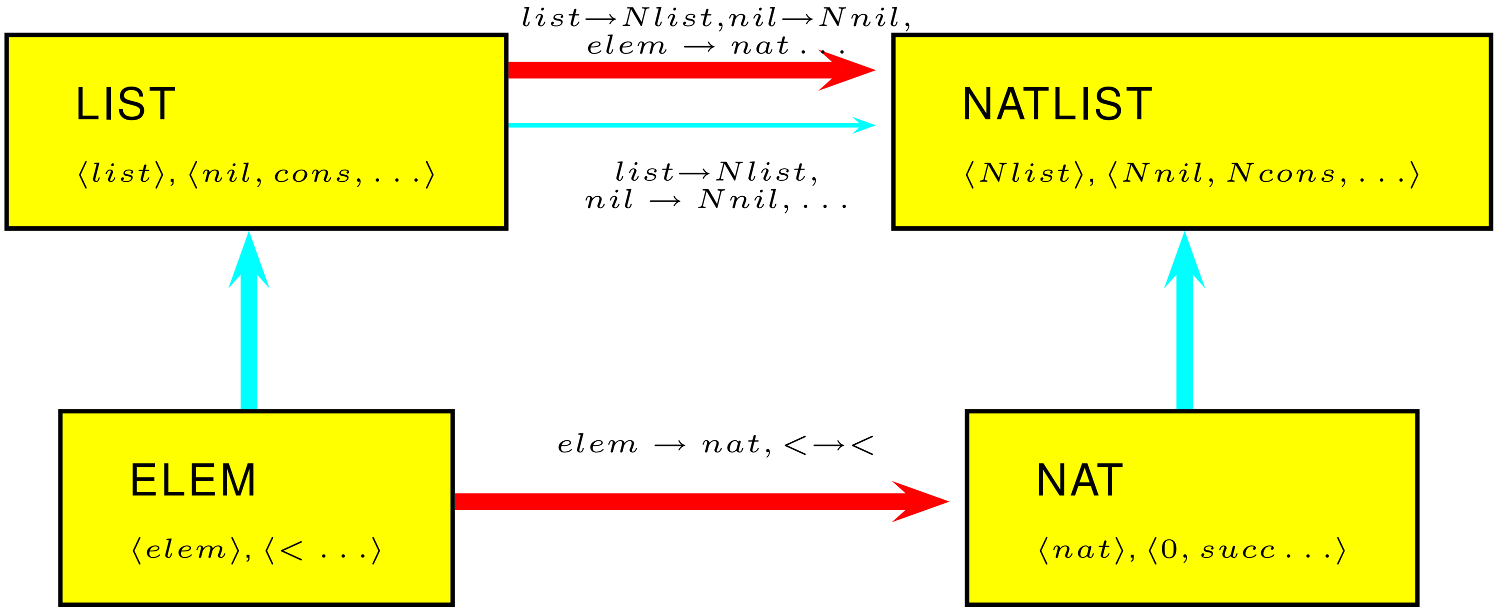


Source: Serge Autexier

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



Development Graphs III

MAYA

Development Graph

=

Structured Logical Content of Specifications

+

Status of proof obligations

(pending, proven, used axioms, . . .)

Structured Specification

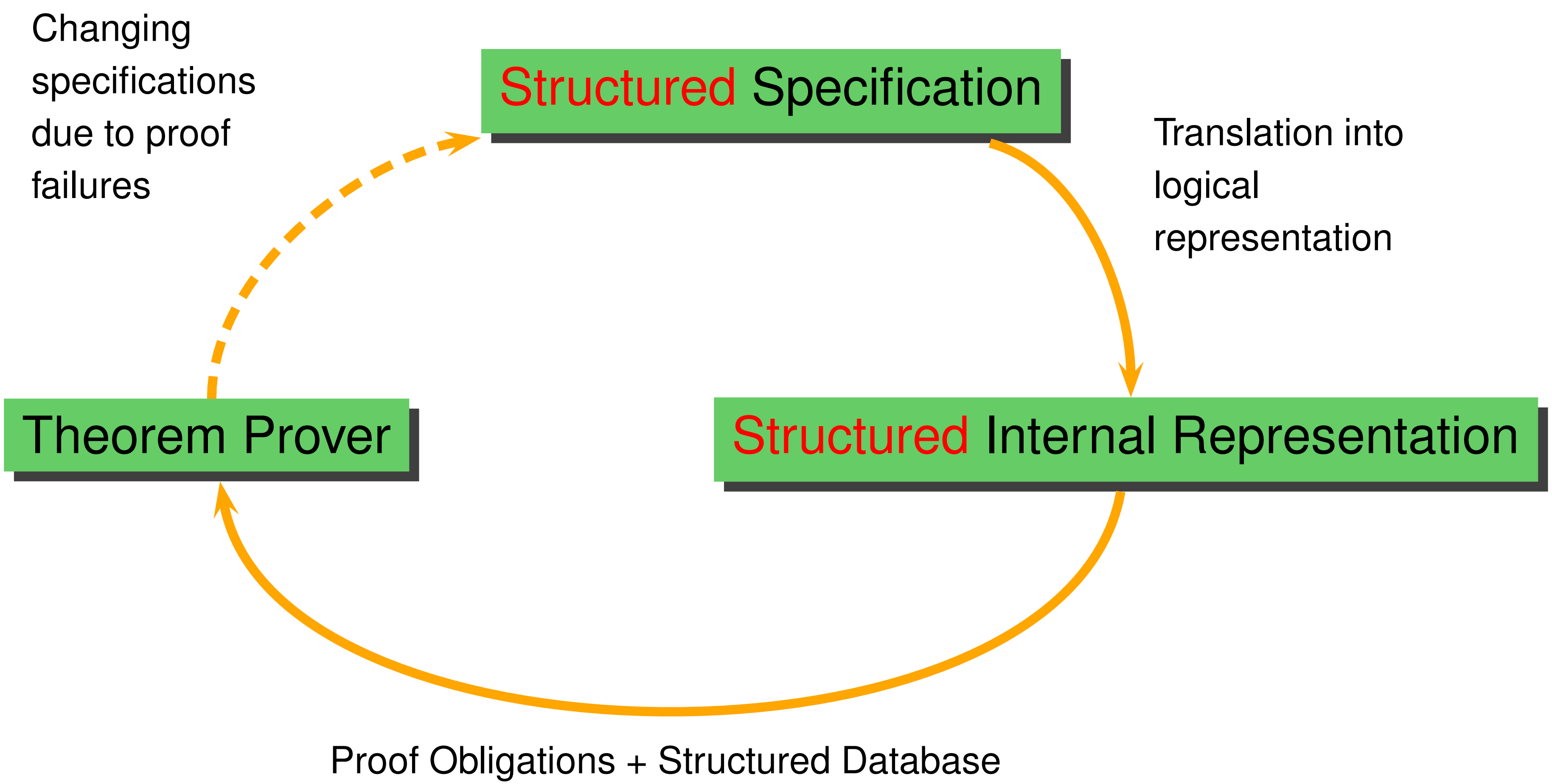
Verification +

Management

Source: Serge Autexier

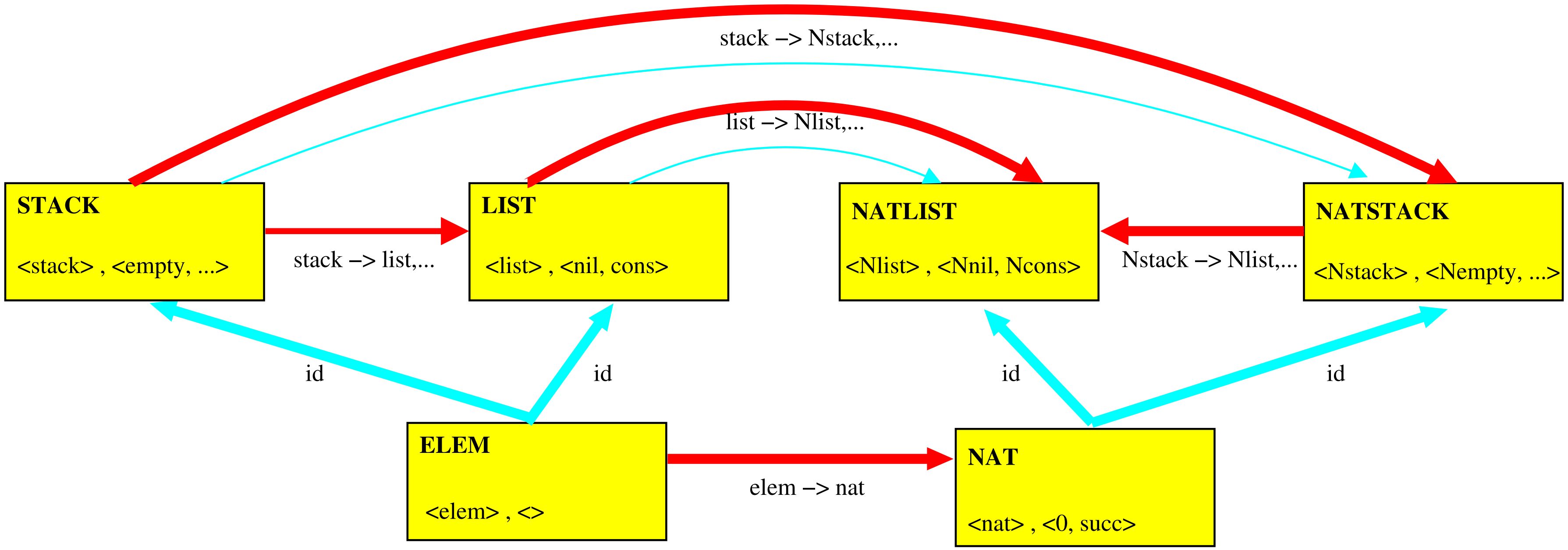
Context

Evolutionary Formal Software Development



Source: Serge Autexier

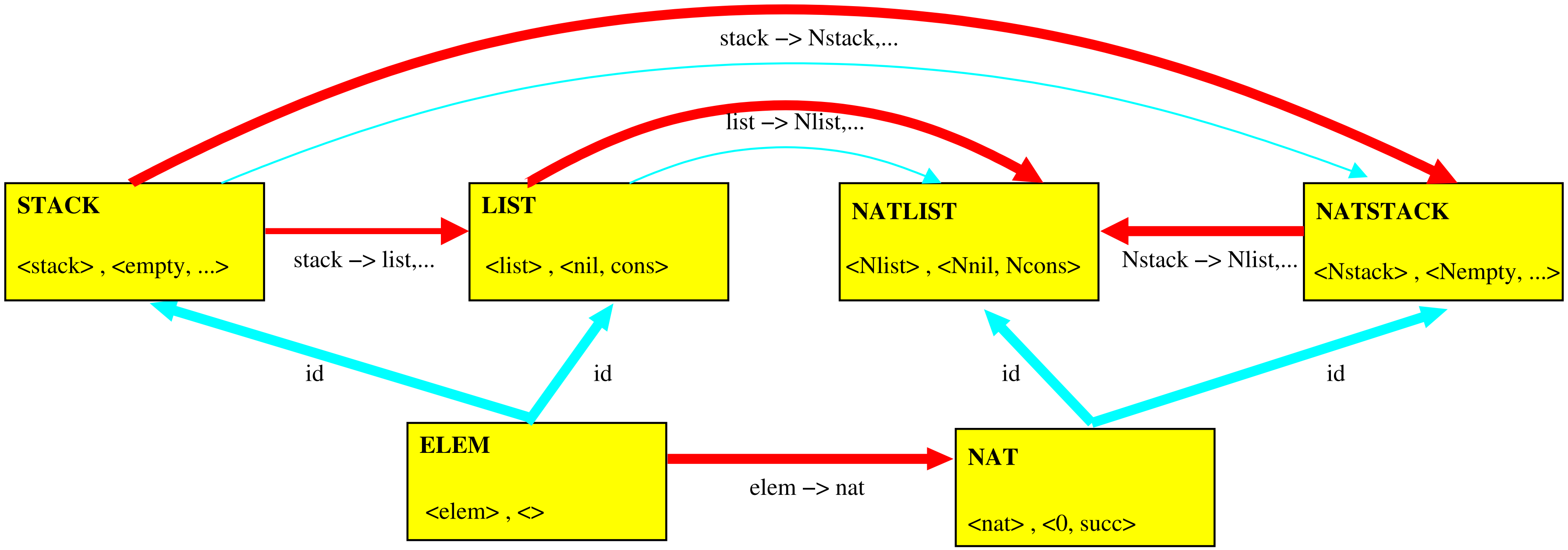
Exploiting the Graph Structure



Source: Serge Autexier

Exploiting the Graph Structure

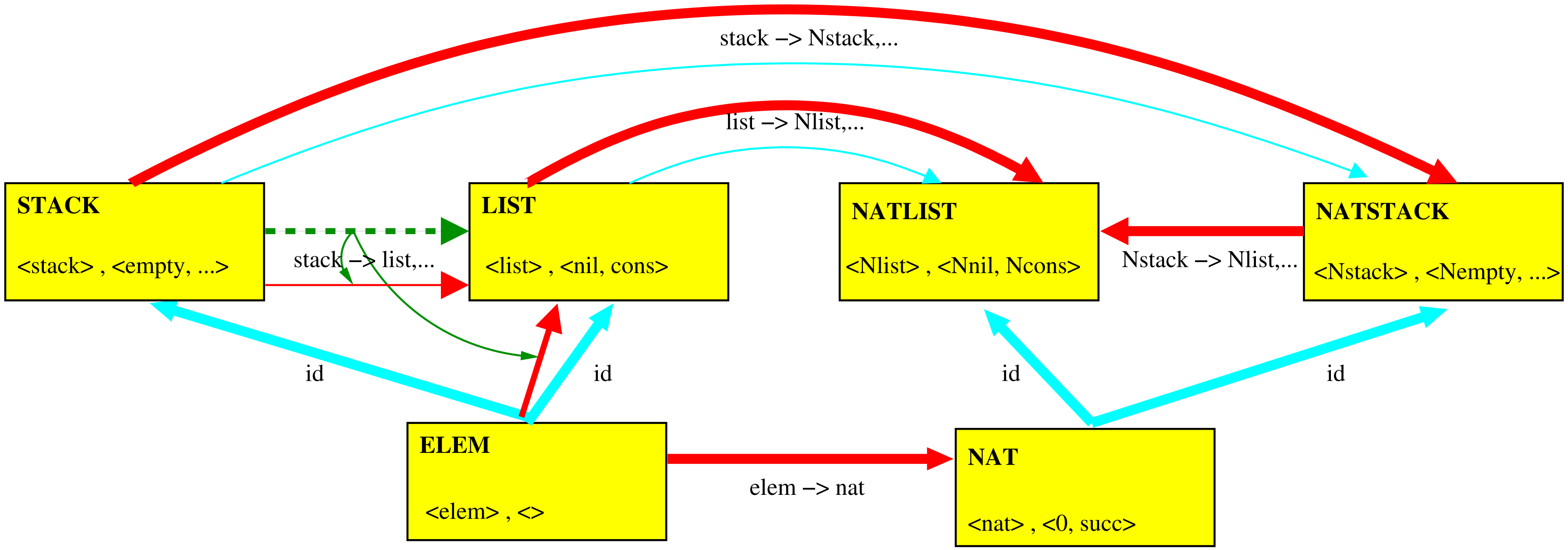
MAYA



33 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

Exploiting the Graph Structure

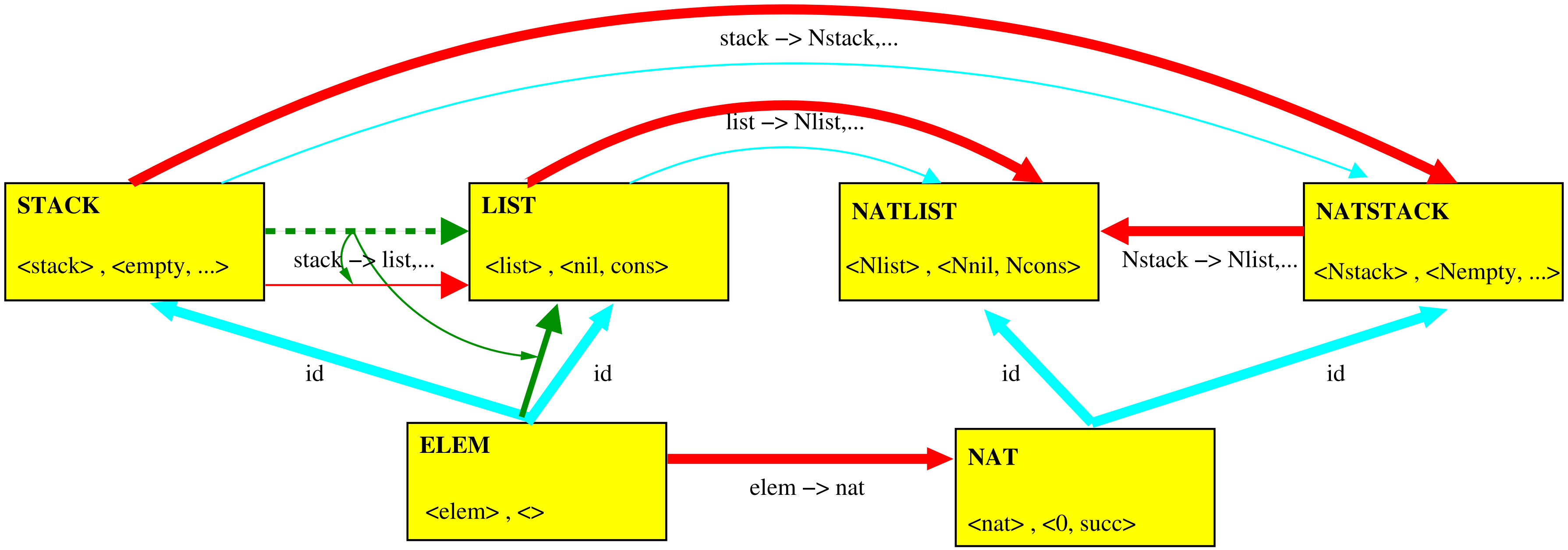


Decomposition of Global Links into Local Links

Source: Serge Autexier

Exploiting the Graph Structure

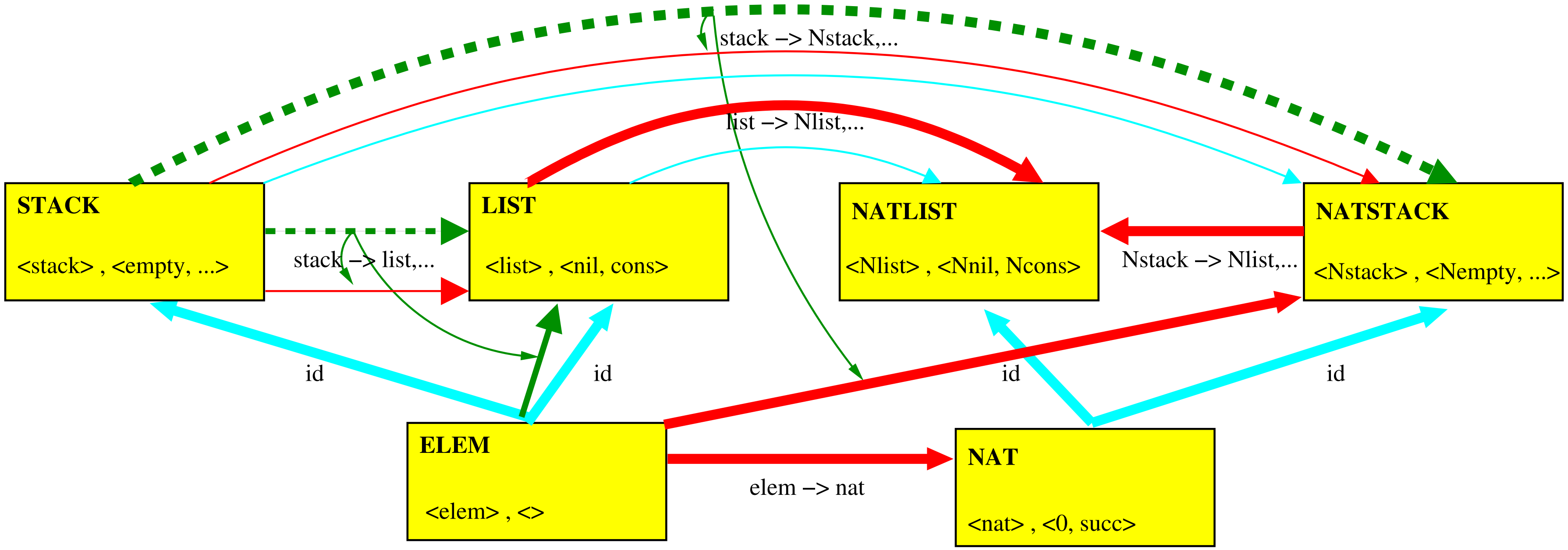
MAYA



Source: Serge Autexier

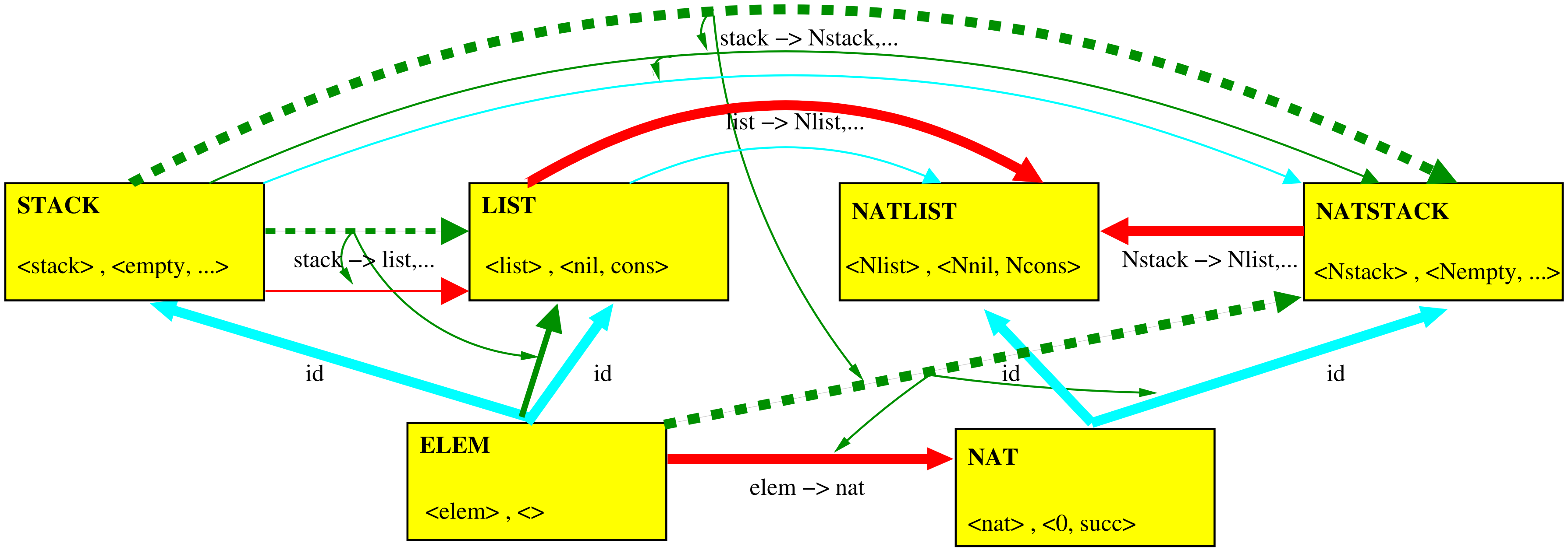
Exploiting the Graph Structure

MAYA



Source: Serge Autexier

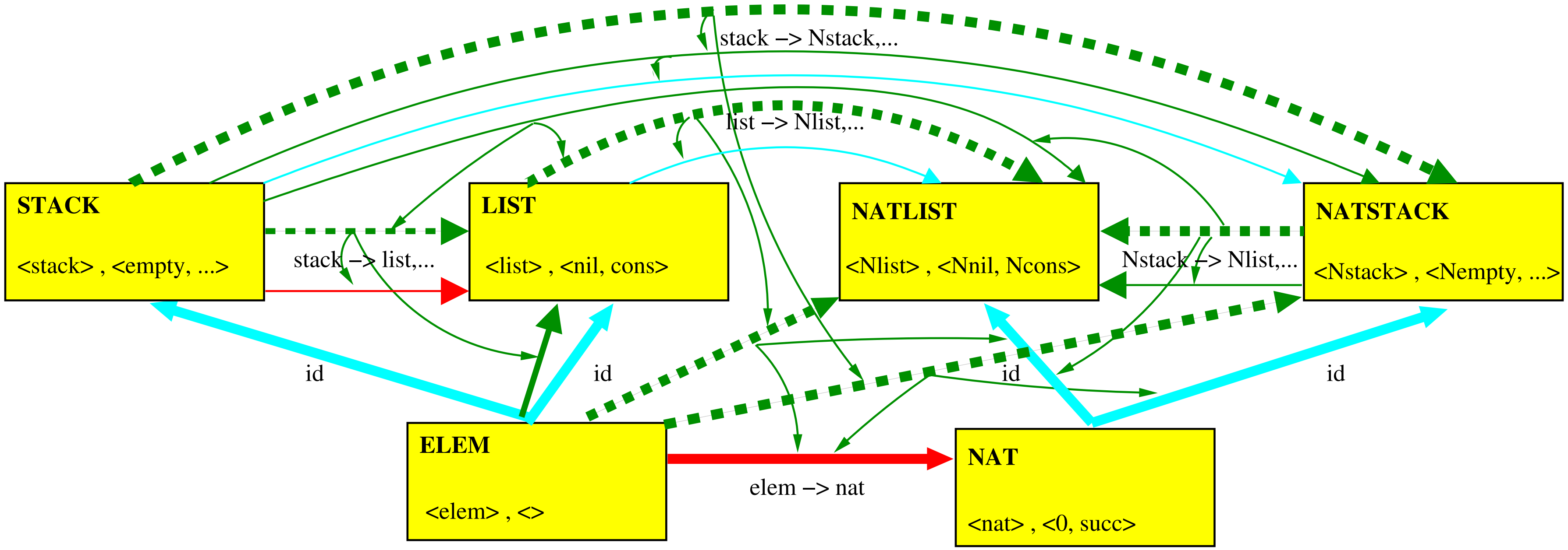
Exploiting the Graph Structure



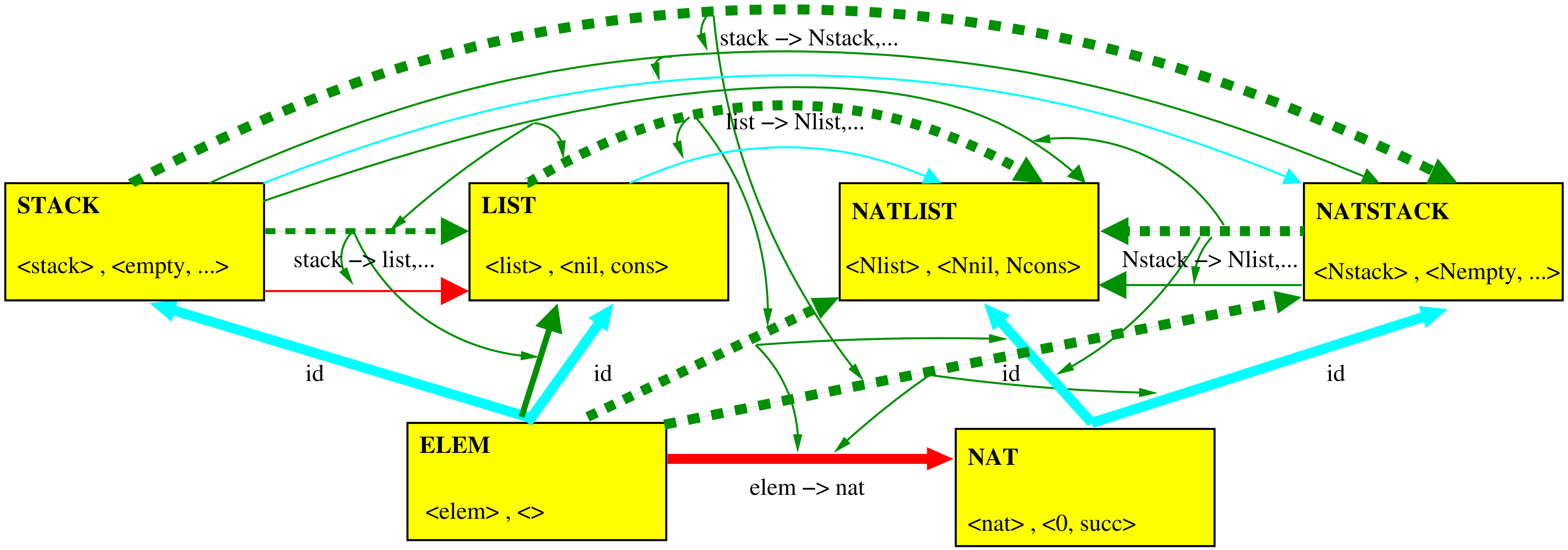
Subsumption of Links by Paths

Exploiting the Graph Structure

MAYA



Exploiting the Graph Structure



8 Proof obligations:

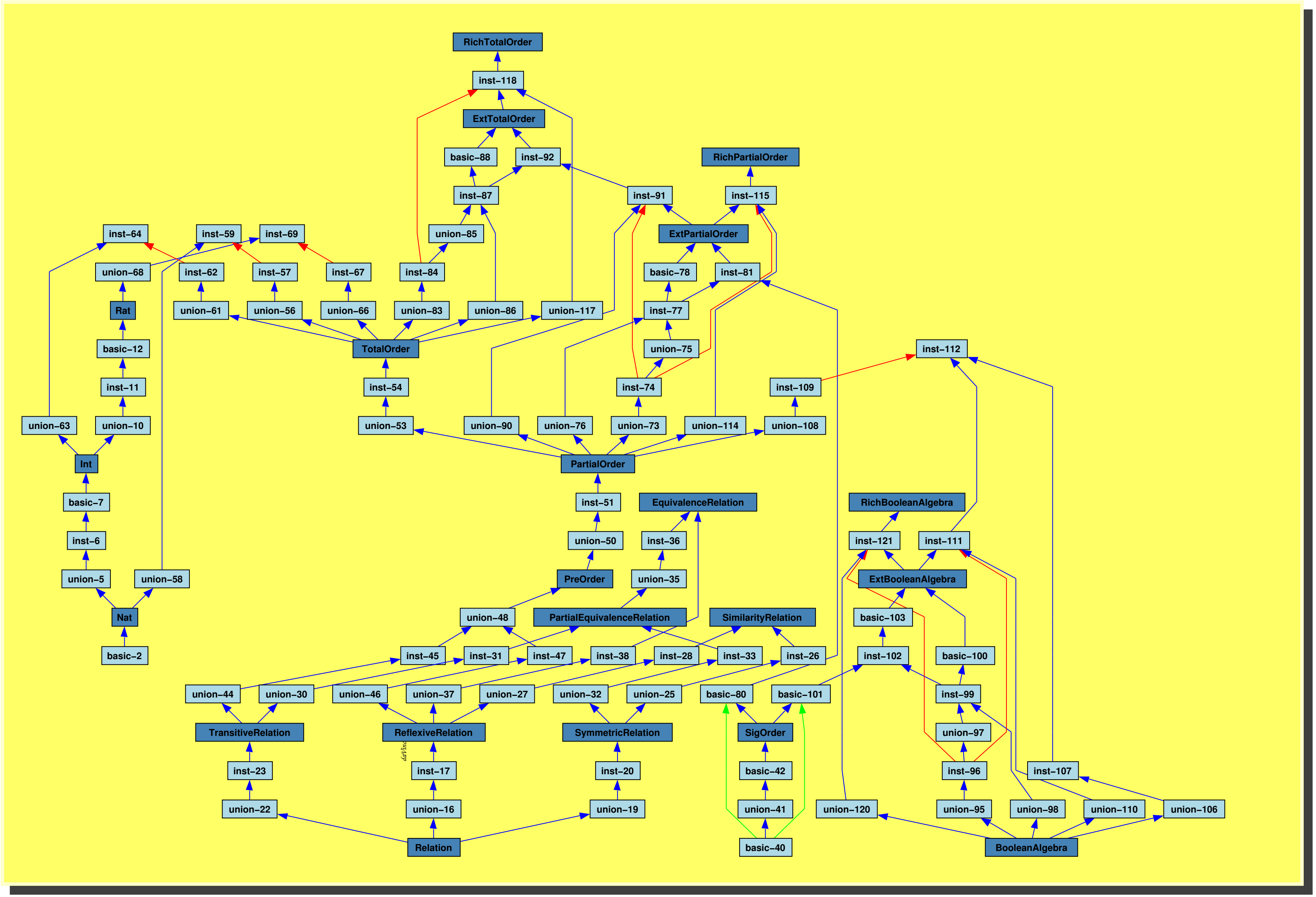
· Local axioms from STACK in LIST · Local axioms from ELEM in NAT

Reduction of $\approx \mathbf{75\%}$ by exploiting graph structure

Source: Serge Autexier

Development Graphs IV

MAYA



Source: Serge Autexier

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

Source: Serge Autexier

Structured Verification

Verification in-the-large

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

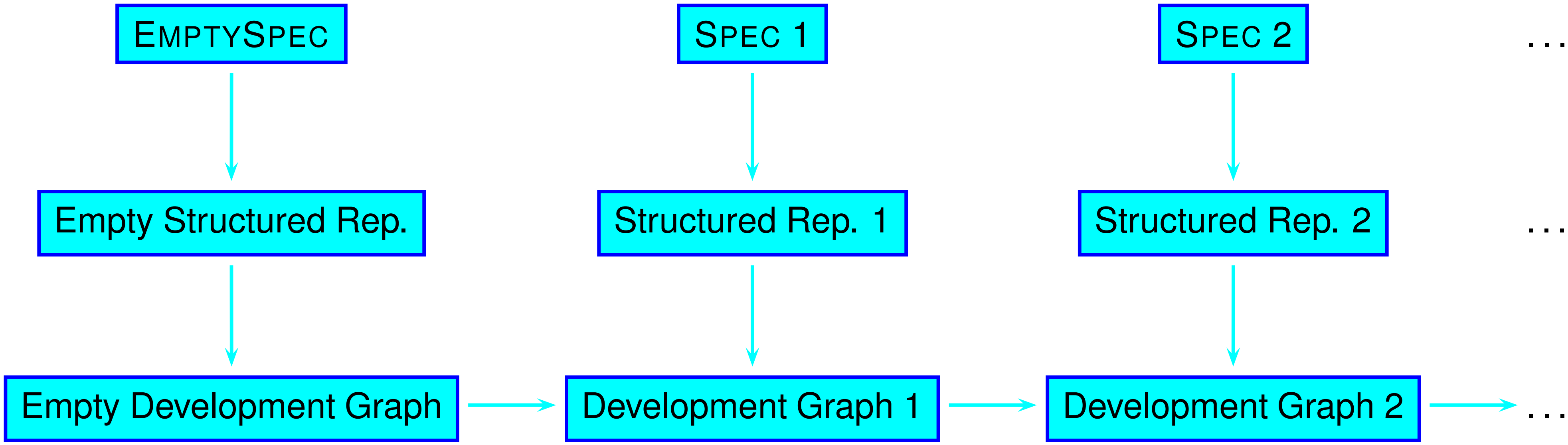
Verification in-the-small

- Remaining proof obligations must be tackled by some theorem prover

Verification Tools

- 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

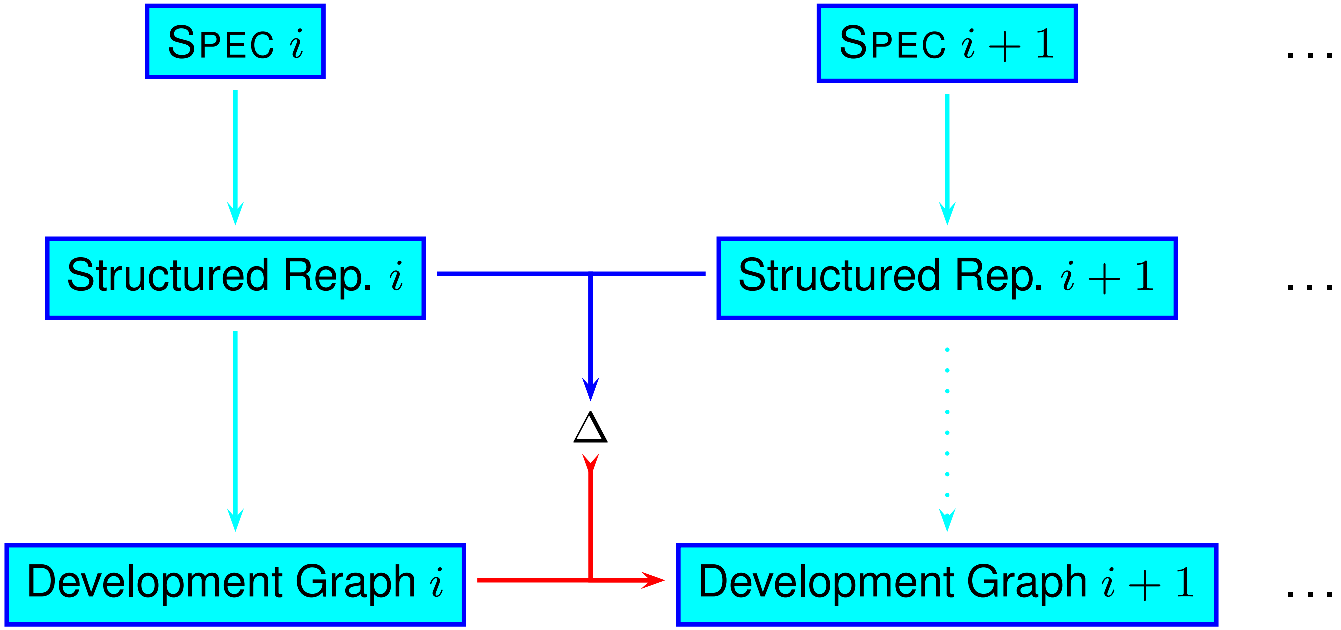
Unwinding Evolutionary Formal Software Development



Development Graph = Structured Representation + Decomposition + Proofs of theorems

Source: Serge Autexier

More Closely...

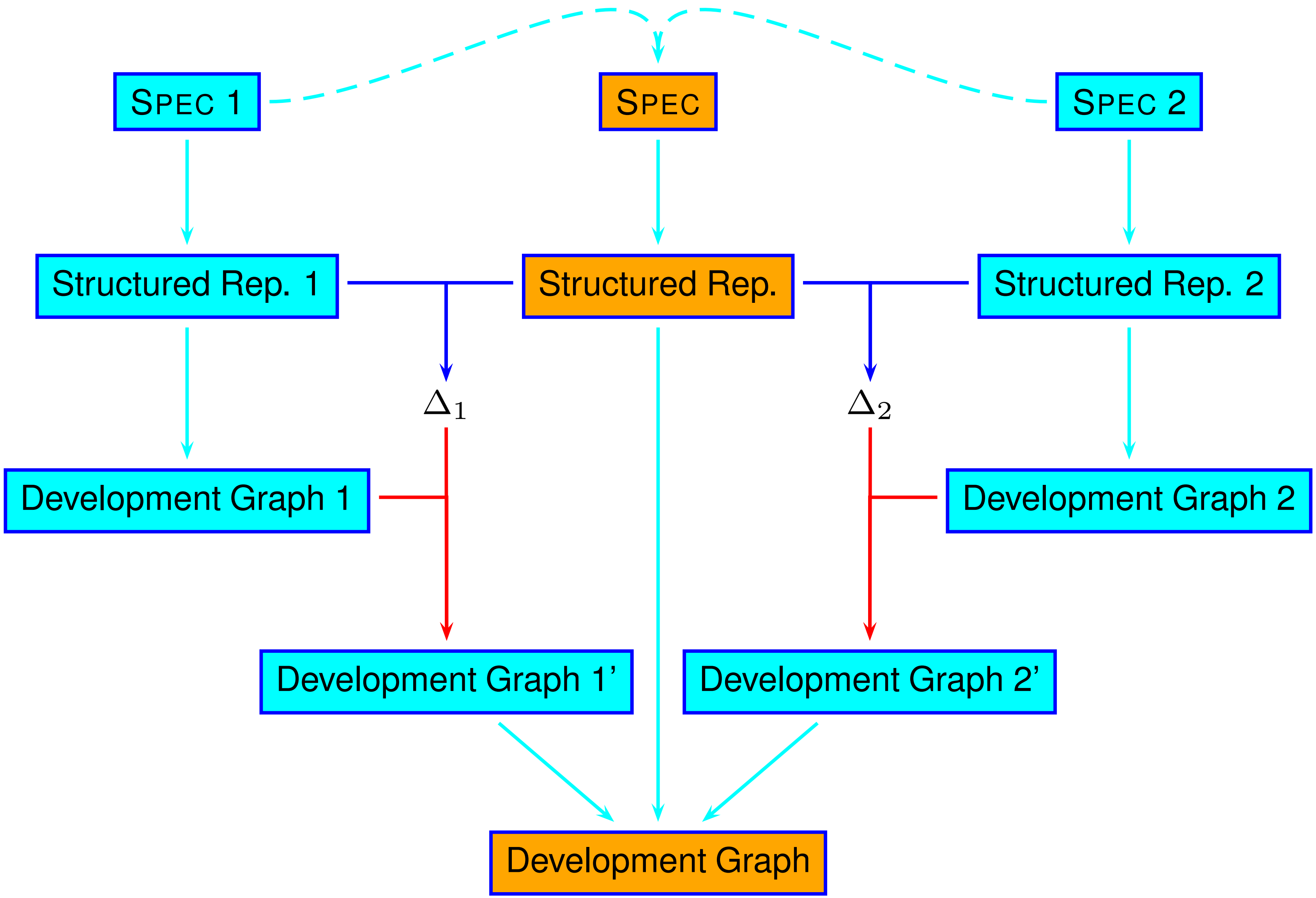


- Δ : Basic operations to adapt development graph
- — : Difference Analysis to compute basic operations
- — : Execution of basic steps followed by strategies guiding verification-in-the large
  - to preserve proofs, link decompositions & link subsumptions
  - to derive new link decomposition & new link subsumptions

Source: Serge Autexier

Distributed Formal Software Developments

MAYA



Source: Serge Autexier

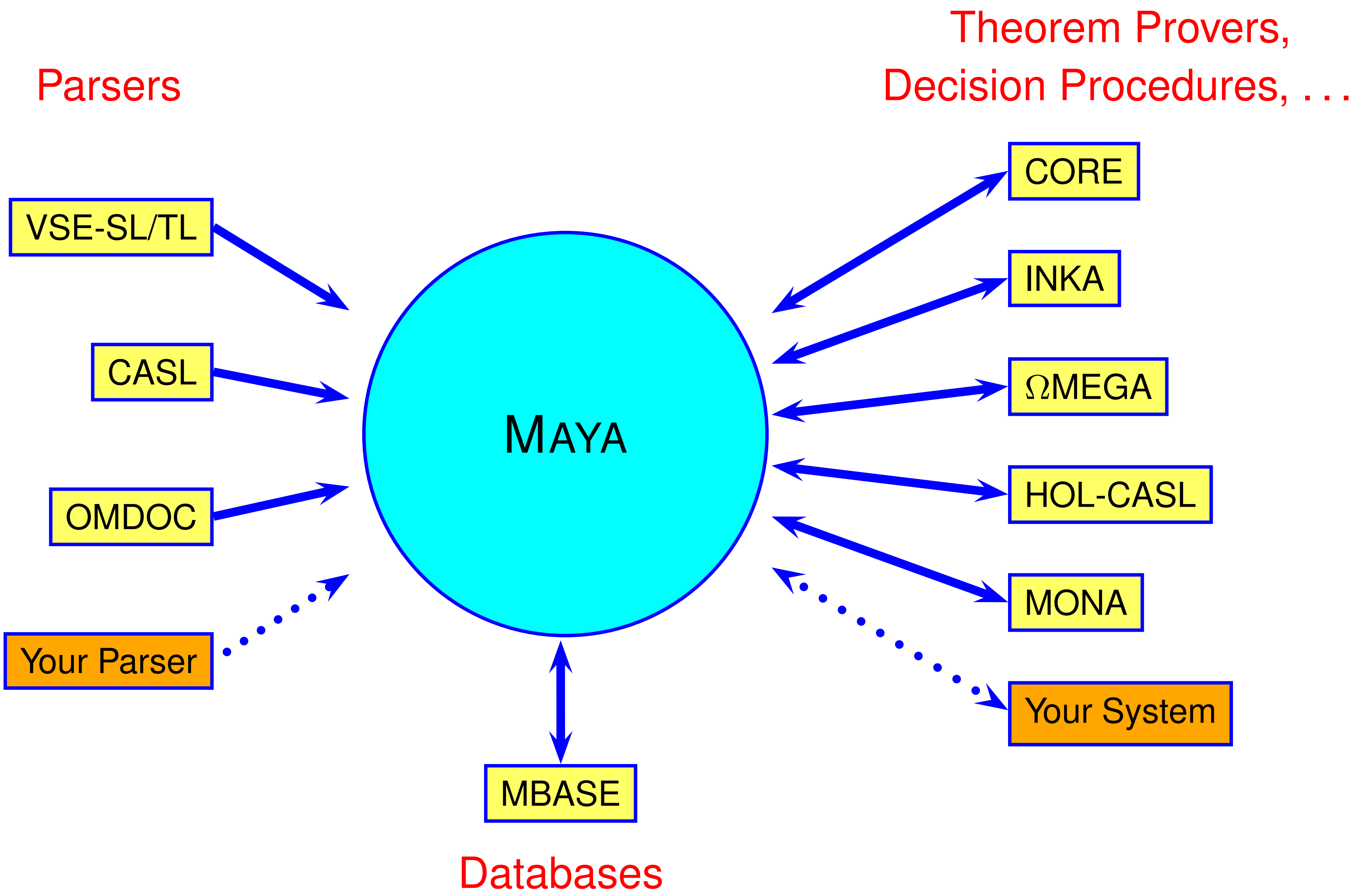
The Development Manager MAYA

MAYA

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

Source: Serge Autexier

MAYA's Social Life



Source: Serge Autexier

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 $\mathcal{L}$
  - $\longrightarrow$: Insertion/Deletion of signature declarations, axioms, prove conjecture
  - $\longleftarrow$: 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

    $\Longrightarrow$ more adequate representation of proof obligations

    $\Longrightarrow$ 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

Source: Serge Autexier

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 of changes to the validity of proofs
  - Dependency analysis + Timestamps
- Uniform interface to theorem provers

Source: Serge Autexier

Ongoing and Future Work

MAYA

- 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 pecialized provers / decision procedures
- Lemmaspeculation exploiting graph structure
- Reuse of proofs

More Information...