UNIVERSIDAD
POLITECNICA
DE VALENCIA

Verification and Certification Using Rewriting Logic

Santiago Escobar

Technical University of Valencia

Outline

Verification and Certification in Rewriting Logic

1. Web Verdi-M: a rule-based Web verification system
2. Maude-NPA: a crypto protocol analyzer
3. Automated Certification of Java Source Code in Maude

Why rewriting logic (Maude)?

1. Models and formal specification are easily written in Maude (Simplicity, Expressiveness, and Performance)
2. Maude provides rewriting modulo associativity, commutativity and identity
3. Differentiation between concurrent and deterministic fragments of a model, differentiation between non-terminating and terminating fragments of a model
4. It provides order-sorted specifications
5. It provides an infrastructure for formal analysis and verification (including a search command, a LTL model checker, a theorem prover, etc.
6. Reflection (meta-modeling, symbolic execution, etc.)
7. Models of computation (λ-calculi, π-calculus, petri nets, CCS), Programming languages (Java, Haskell, Prolog), Distributed algorithms and systems (real-time, probabilistic), Biological systems in Maude

Web Verdi-M

(A rule-based Web verification system)

María Alpuente (Universidad Politecnica de Valencia, Spain)
Demis Ballis (Università di Udine, Italy)
Moreno Falaschi (Università di Siena, Italy)
Pedro Ojeda (Universidad Politécnica de Valencia, Spain)
Daniel Romero (Universidad Nacional de Río Cuarto, Argentina)

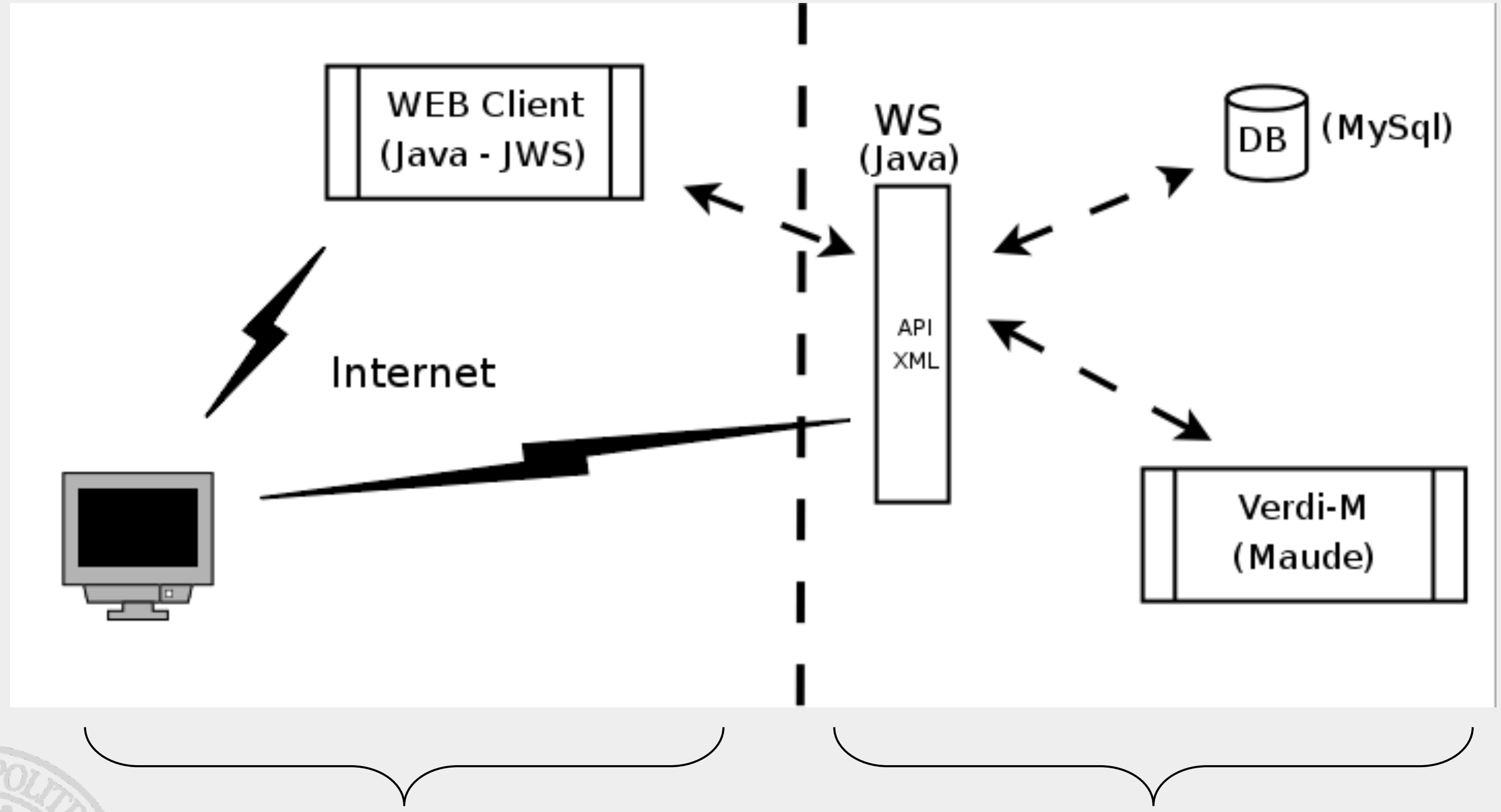
Web Verdi-M : Goals

- Web Service and web client for verification of web sites w.r.t. an intended behavior.
- Optimization and repairing support.

Intended Behavior

- We have a specification language to verify if a web site is correct w.r.t. its specification.
- This specification contains rules:
  - Correctness rules (for detecting incorrect web pages). These rules have the following form:
    - member(name(X) , surname(Z)) -> error : X == Z
  - Completeness rules (for detecting information incomplete and/or missing web pages). These rules have the following form:
    - hpage(status(Professor)) -> #hpage(#status(#Professor) , teaching( )) < A >

Web Verdi-M: Architecture



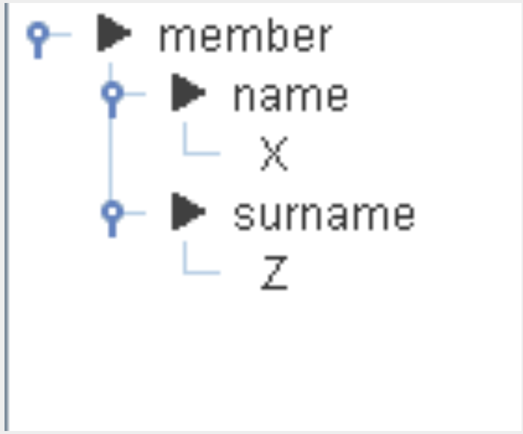
Front-end

Back-end

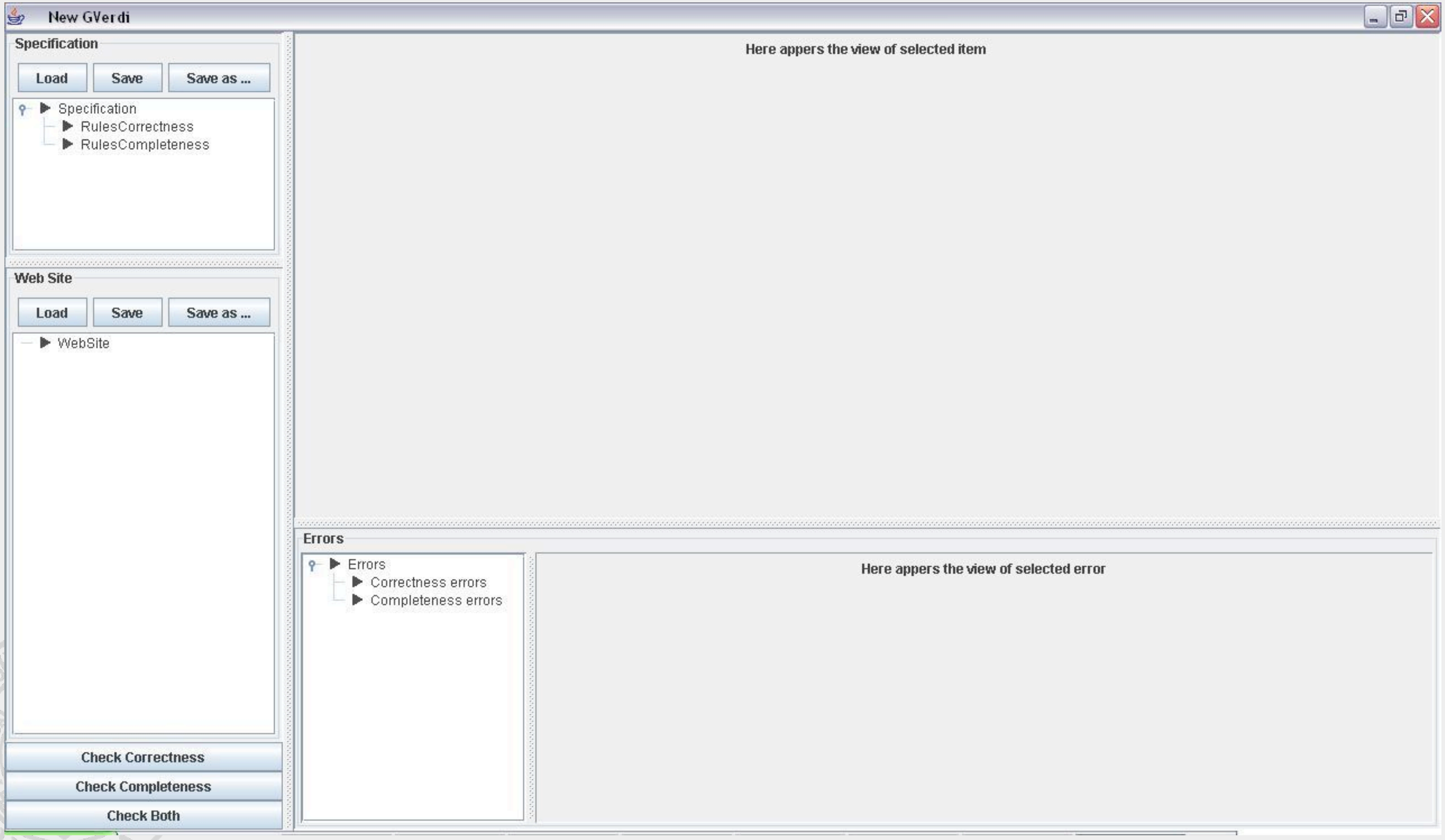
Graphical interface

- The graphical interface is based on tree directories.

- Example:
  - member(name(X),surname(Z))

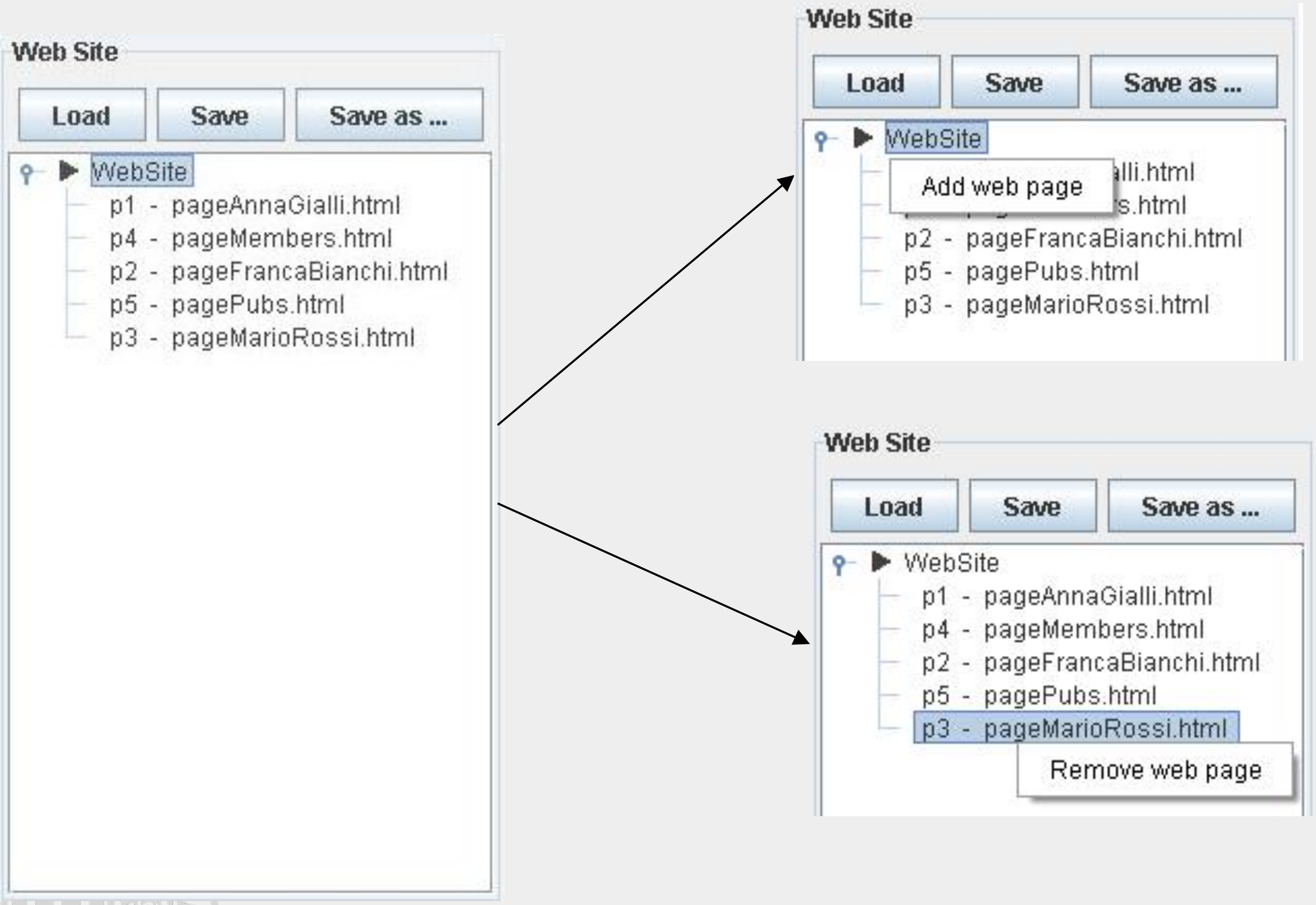


Main Window

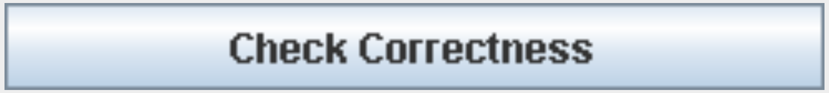


New GVerdi
Specification
Load
Save
Save as ...
Specification
RulesCorrectness
RulesCompleteness
Web Site
Load
Save
Save as ...
WebSite
Check Correctness
Check Completeness
Check Both
Here appers the view of selected item
Errors
Errors
Correctness errors
Completeness errors
Here appers the view of selected error

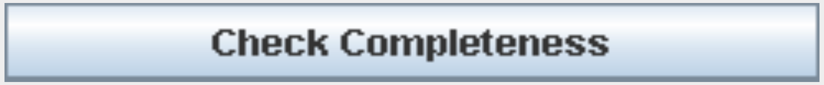
Web Site Panel



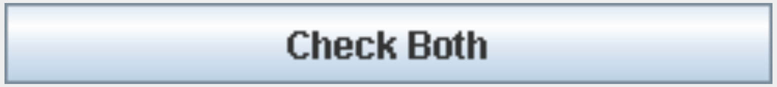
Check Rules - Options



- Check via the web service, the correctness rules loaded.

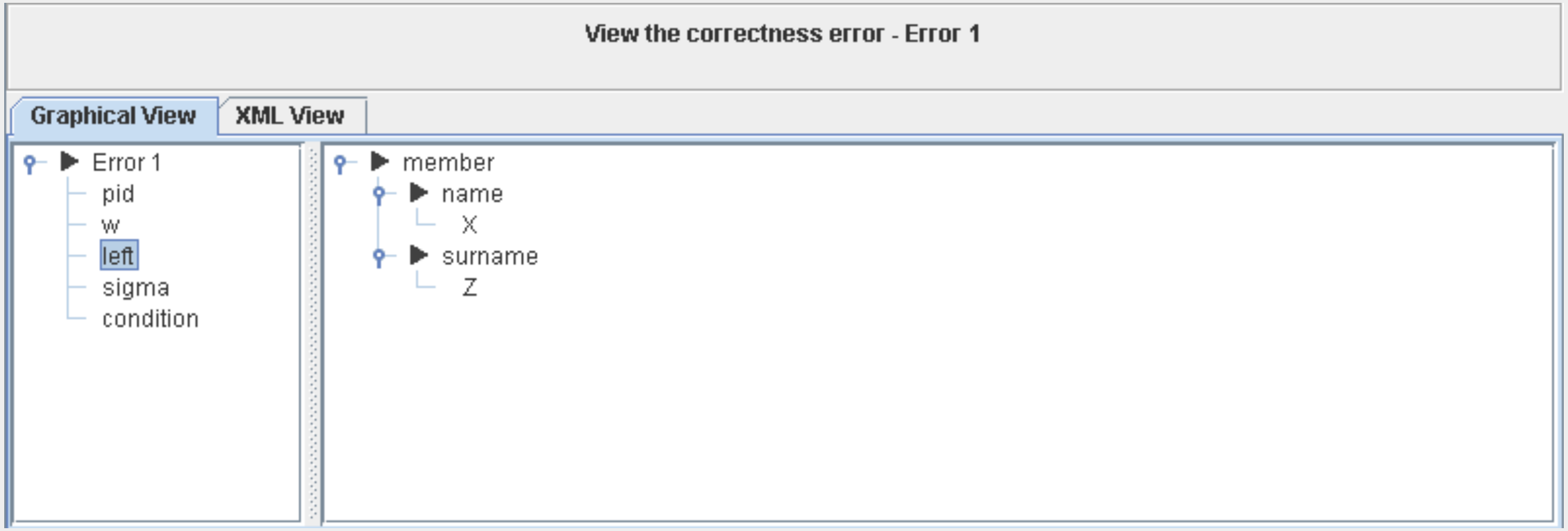


- Check via the web service, the completeness rules loaded.



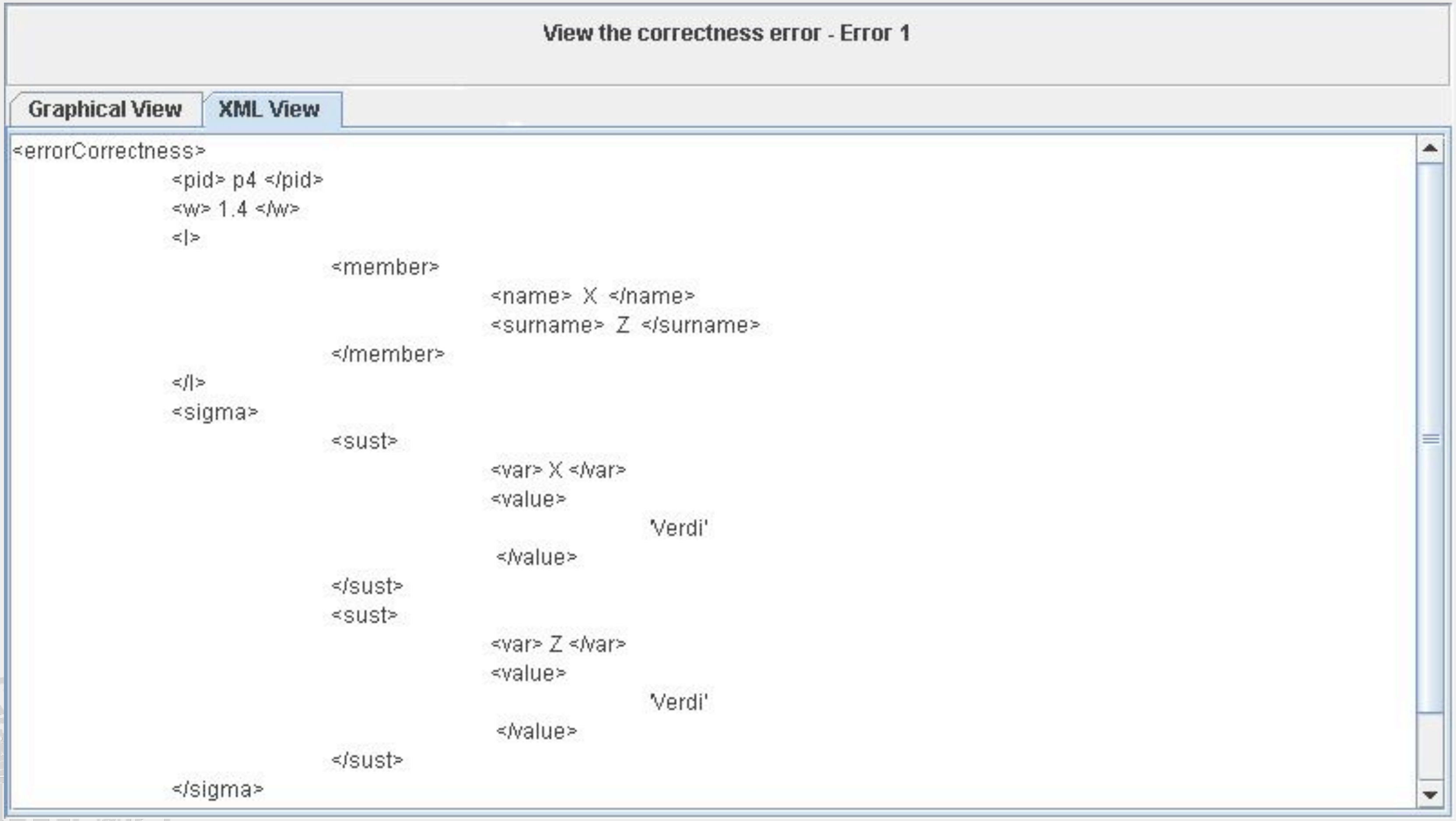
- Check via the web service, the correctness and completeness rules loaded.

Error Views – Correctness(1)



View the correctness error - Error 1

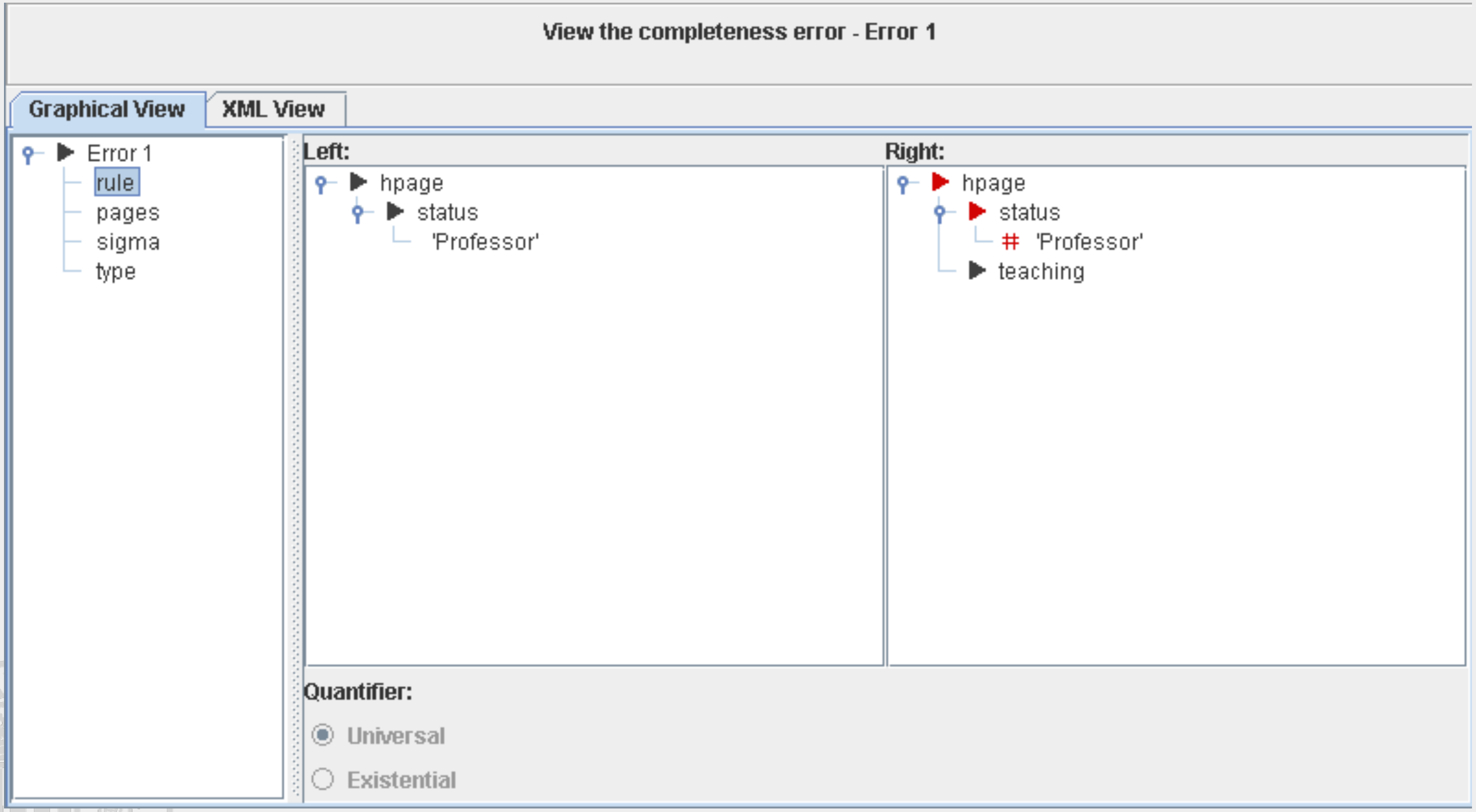
Error Views – Correctness(2)



View the correctness error - Error 1

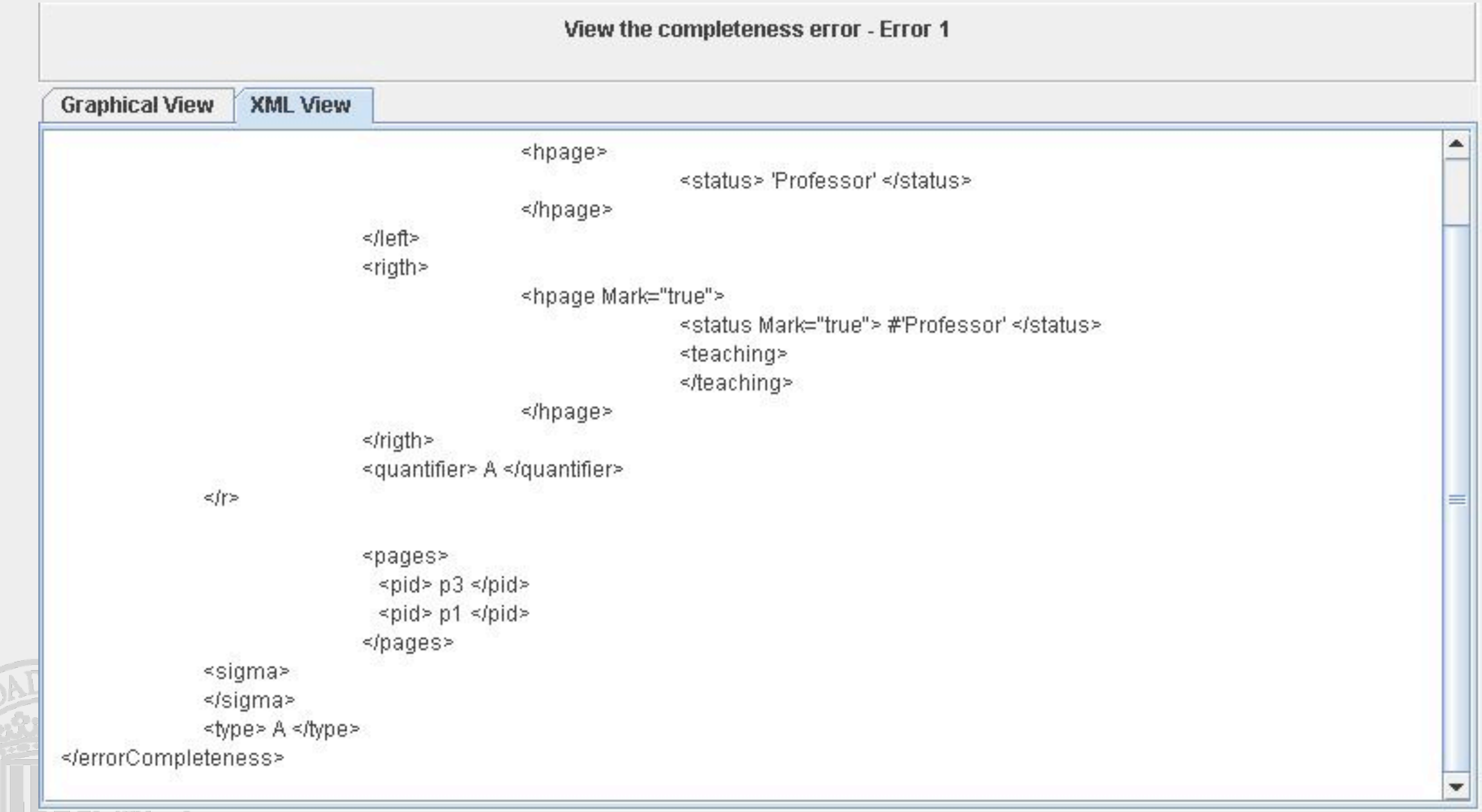
```
<errorCorrectness>
            <pid> p4 </pid>
            <w> 1.4 </w>
            <l>
                        <member>
                                    <name> X </name>
                                    <surname> Z </surname>
                        </member>
            </l>
            <sigma>
                        <sust>
                                    <var> X </var>
                                    <value>
                                                'Verdi'
                                     </value>
                        </sust>
                        <sust>
                                    <var> Z </var>
                                    <value>
                                                'Verdi'
                                     </value>
                        </sust>
            </sigma>
```

Error Views – Completeness(1)



View the completeness error - Error 1

Error Views – Completeness(2)



View the completeness error - Error 1

```
                        <hpage>
                                <status> 'Professor' </status>
                        </hpage>
                </left>
                <rigth>
                        <hpage Mark="true">
                                <status Mark="true"> #'Professor' </status>
                                <teaching>
                                </teaching>
                        </hpage>
                </rigth>
                <quantifier> A </quantifier>
        </r>

                <pages>
                  <pid> p3 </pid>
                  <pid> p1 </pid>
                </pages>
        <sigma>
        </sigma>
        <type> A </type>
</errorCompleteness>
```

Extensions and Optimizations

- Abstract Verification (via Source-to Source Compression Transformation)
- Ontology reasoning

Maude-NPA: crypto protocol analyzer

Santiago Escobar (Universidad Politécnica de Valencia, Spain)
Catherine Meadows (Naval Research Laboratory, USA)
José Meseguer (University of Illinois at Urbana-Champaign, USA)
Sonia Santiago (Universidad Politécnica de Valencia, Spain)
Carolyn Talcott (SRI International, USA)

General Maude-NPA Goal

- Crypto protocol analysis with the standard free algebra model ("Dolev-Yao") well understood
- Extend standard free algebra model of cryto protocol analysis to deal with algebraic properties:
  1. Encryption-decryption
  2. Diffie Hellman
  3. Exclusive-or, etc.
- Provide tool than can be used to reason about protocols with these algebraic properties in the unbounded session model
- Provide graphical interface for analysis, interaction and validation of crypto protocols

Overview of Maude-NPA

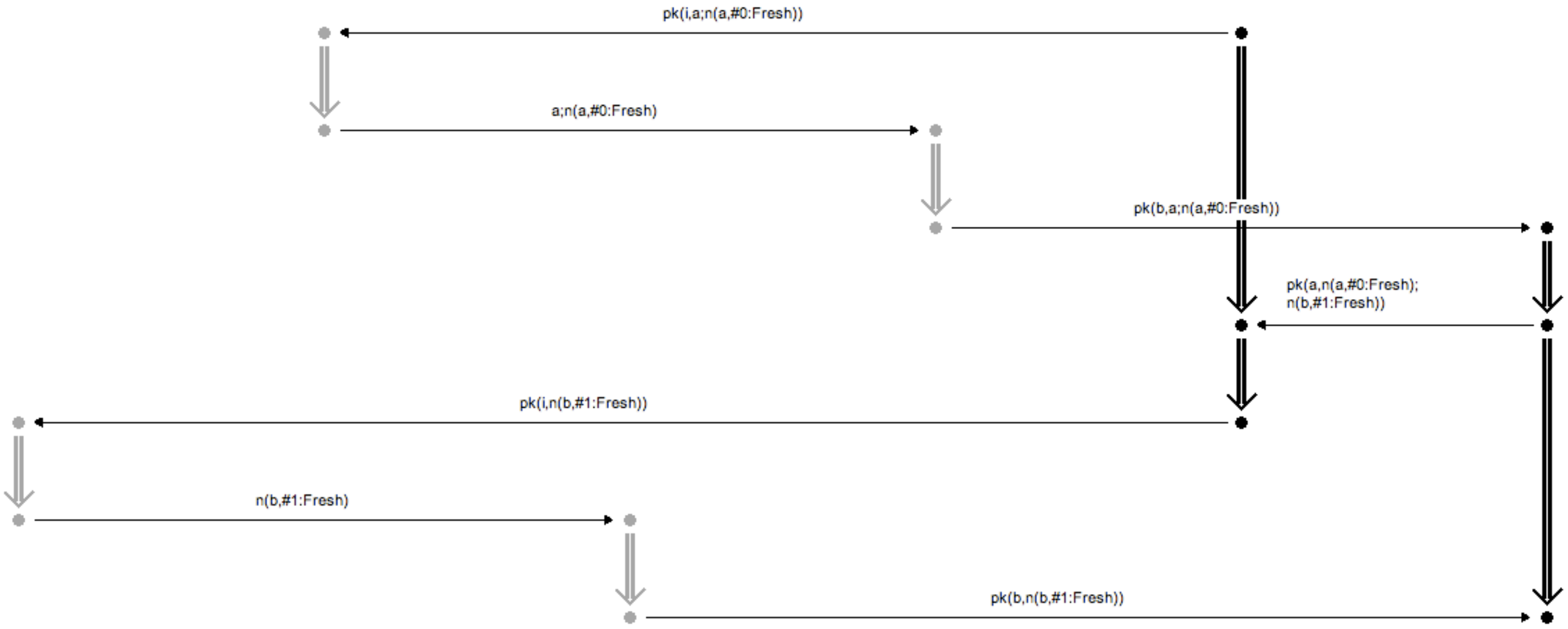
- Use rewriting logic as general theoretical framework
  - rewrite rules are obtained from strands
  - algebraic identities as equational properties and axioms
- Use narrowing modulo equational theories in two ways
  - as a symbolic reachability analysis method
  - as an extensible equational unification method
- Combine with state reduction techniques of NRL Protocol Analyzer (grammars, optimizations, etc.)
- Implement in Maude programming environment
  - Rewriting logic gives us theoretical framework and understanding
  - Maude implementation gives us tool support
- Define graphical interface within Maude and its associated frameworks for graphical interaction (IOP, IMaude and JLamba)

Maude-NPA

- A tool to find or prove the absence of attacks using backwards search
- Analyzes infinite state systems
  - Active intruder
  - No abstraction or approximation of nonces
  - Unbounded number of sessions
- Intruder and honest protocol transitions represented using variant of strand space model
- Algebraic identities such as exponentiation and Encryption/Decryption cancellation identities included
- Uses modified strand space model
- Each local execution and each intruder action represented by a strand, plus a marker denoting the current state
  - Searches backwards through strands from final state.
  - Set of rewrite rules governs how search is conducted
  - Sensitive to past and future
- Grammars and other optimizations used to prevent infinite loops and avoid some transitions

How protocols are specified in Maude-NPA

- Represent protocols and intruder actions using strands
  - Strands may contain variables, except for terms of type *Fresh*, which are always constant (used by nonces)
  - Strand annotated with fresh terms generated by principal executing strands
  - :: r :: [pke(B, n (A, r); A)$^{+}$, pke(A, n(A,r); NB)$^{-}$, pke(B,NB)$^{+}$]



Diffie-Hellman Protocol

- Protocol

  A --> B: A ; B ; exp(g, N_A)

  B --> A: A ; B ; exp(g, N_A)

  A --> B: enc(exp(exp(g, N_B) , N_A), secret(A,B))

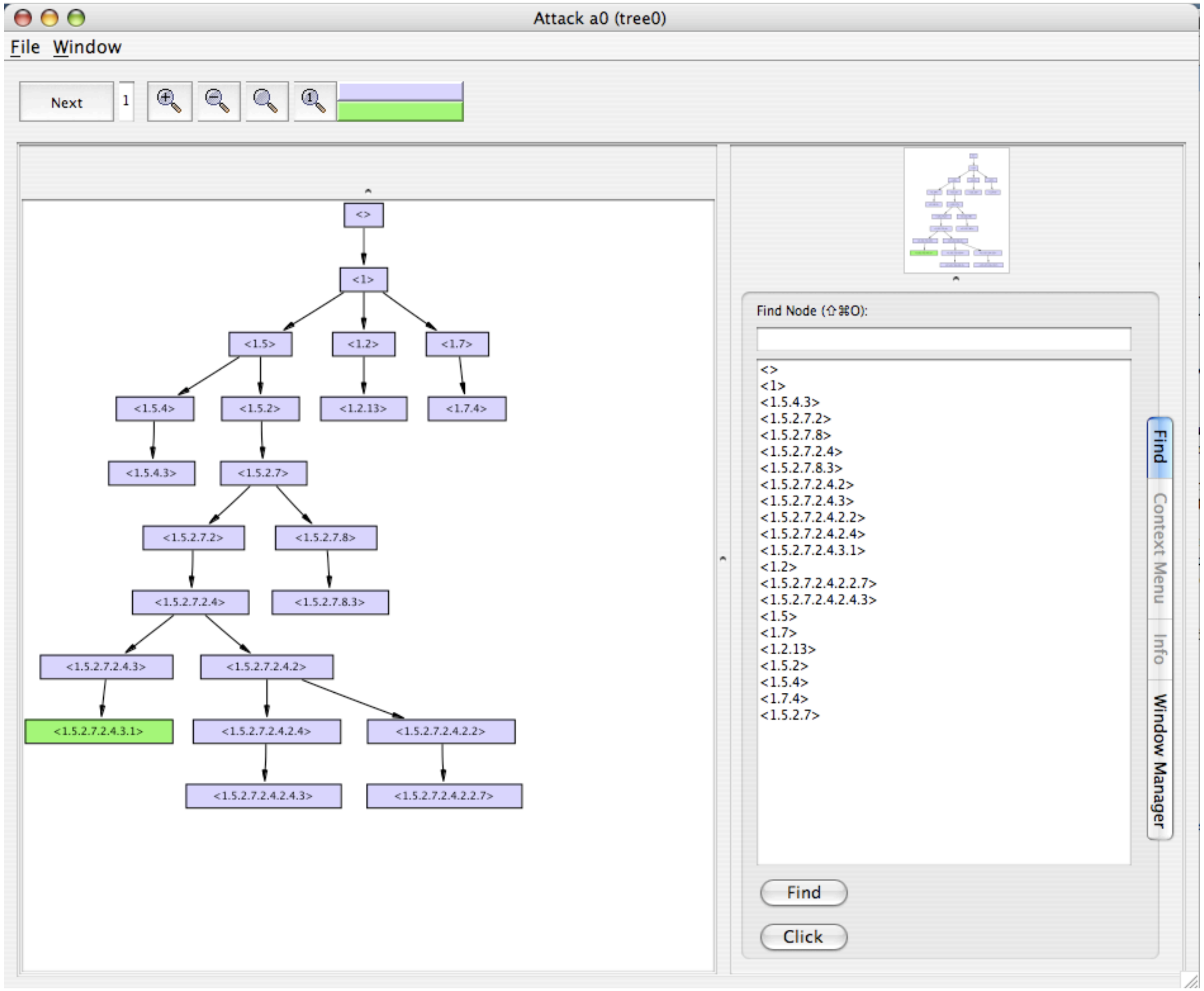
- Equational Theory Algebraic properties

  $B = \{ (X * Y) * Z = X * (Y * Z), (X * Y) = Y * X\}$

  $\Delta = \{ exp(exp(W, Y), Z) = exp(W, Y * Z) \}$

Can B in a session apparently in A without A engaging in the corresponding session?

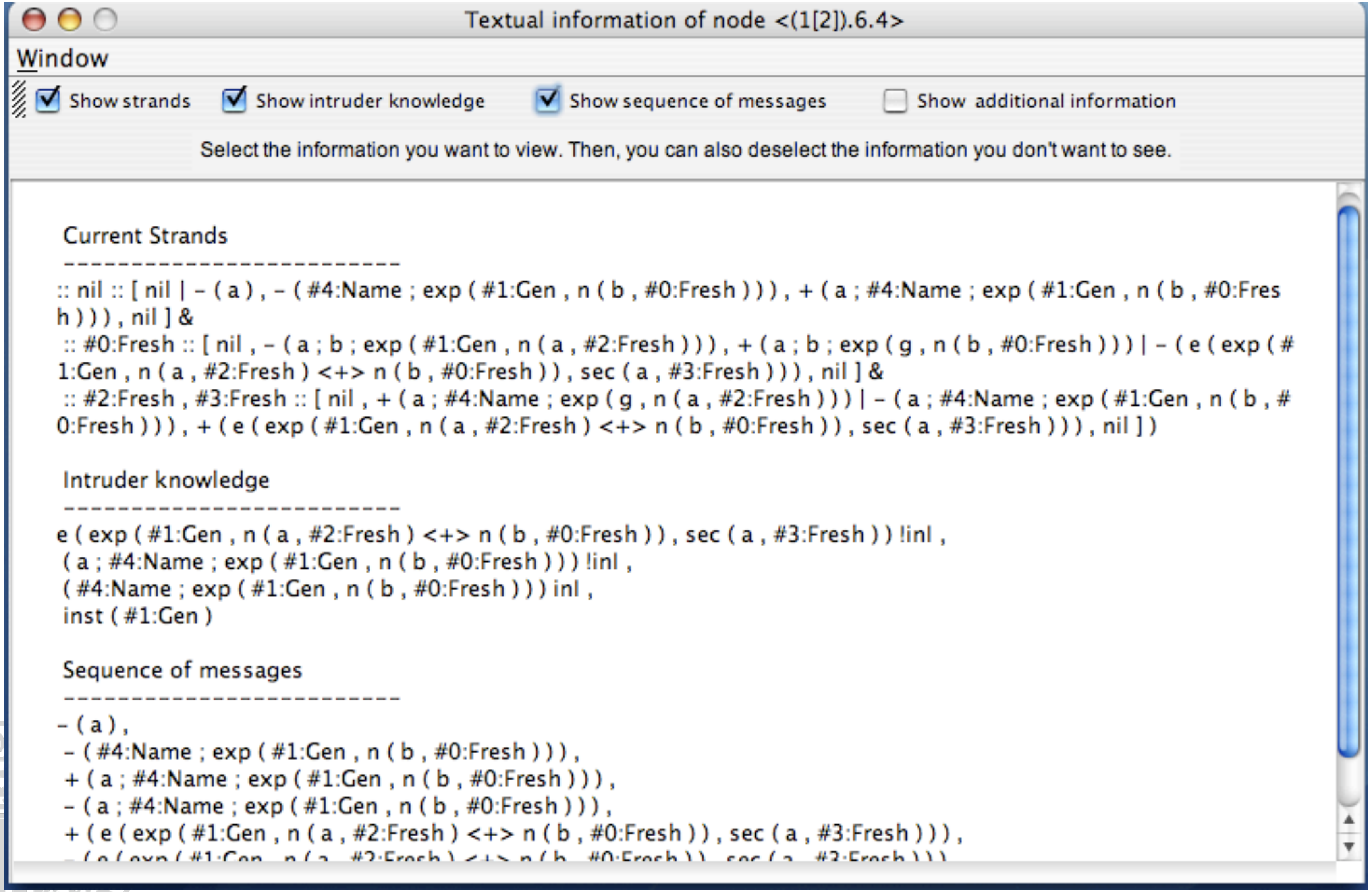
Main Window



Find Node (⇧⌘O):

```
<>
<1>
<1.5.4.3>
<1.5.2.7.2>
<1.5.2.7.8>
<1.5.2.7.2.4>
<1.5.2.7.8.3>
<1.5.2.7.2.4.2>
<1.5.2.7.2.4.3>
<1.5.2.7.2.4.2.2>
<1.5.2.7.2.4.2.4>
<1.5.2.7.2.4.3.1>
<1.2>
<1.5.2.7.2.4.2.2.7>
<1.5.2.7.2.4.2.4.3>
<1.5>
<1.7>
<1.2.13>
<1.5.2>
<1.5.4>
<1.7.4>
<1.5.2.7>
```

State info

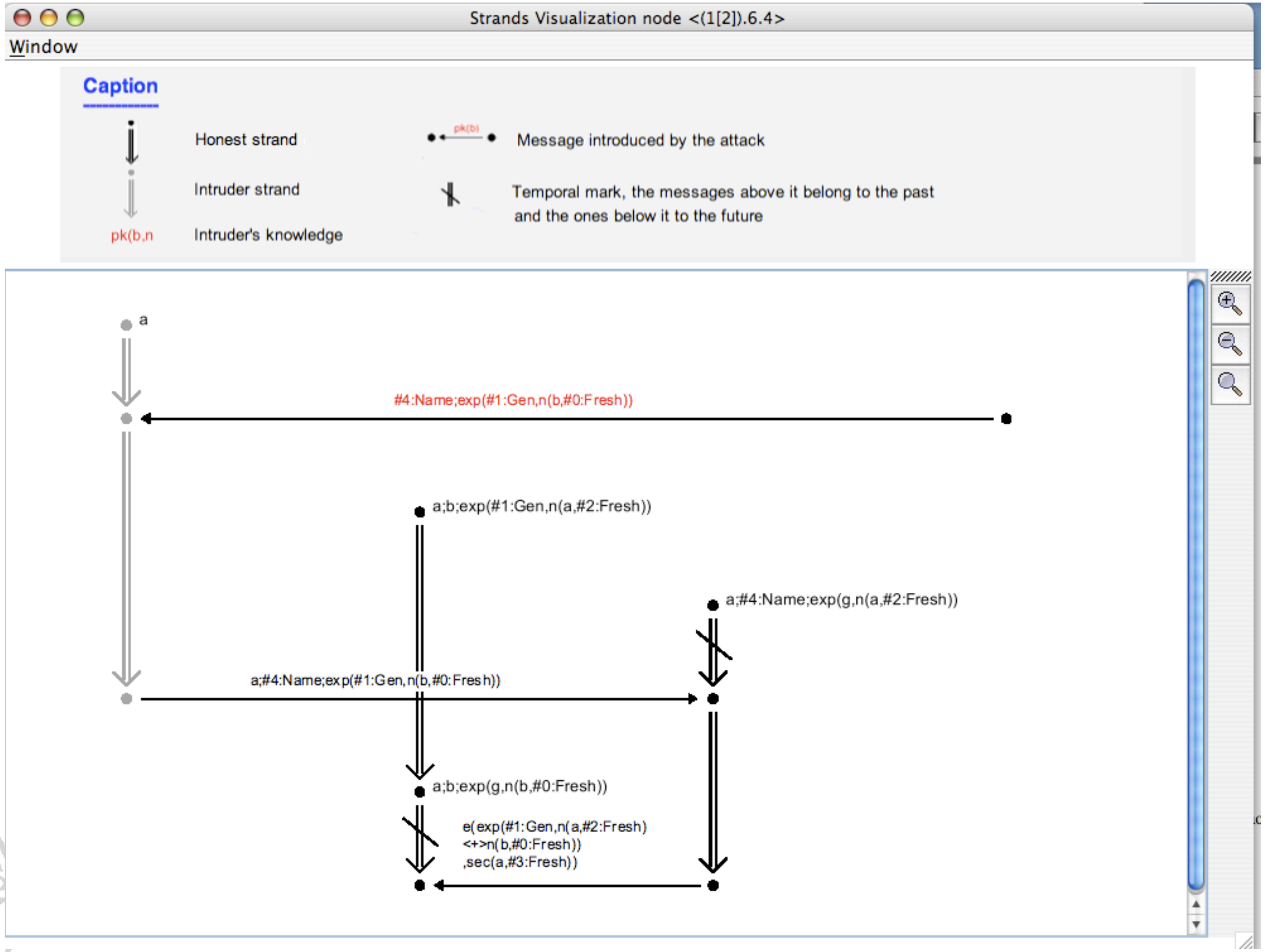


☑ Show strands ☑ Show intruder knowledge ☑ Show sequence of messages □ Show additional information

Select the information you want to view. Then, you can also deselect the information you don't want to see.

```
 Current Strands
 --------------------------
:: nil :: [ nil | - ( a ) , - ( #4:Name ; exp ( #1:Gen , n ( b , #0:Fresh ) ) ) , + ( a ; #4:Name ; exp ( #1:Gen , n ( b , #0:Fres
h ) ) ) , nil ] &
 :: #0:Fresh :: [ nil , - ( a ; b ; exp ( #1:Gen , n ( a , #2:Fresh ) ) ) , + ( a ; b ; exp ( g , n ( b , #0:Fresh ) ) ) | - ( e ( exp ( #
1:Gen , n ( a , #2:Fresh ) <+> n ( b , #0:Fresh ) ) , sec ( a , #3:Fresh ) ) ) , nil ] &
 :: #2:Fresh , #3:Fresh :: [ nil , + ( a ; #4:Name ; exp ( g , n ( a , #2:Fresh ) ) ) | - ( a ; #4:Name ; exp ( #1:Gen , n ( b , #
0:Fresh ) ) ) , + ( e ( exp ( #1:Gen , n ( a , #2:Fresh ) <+> n ( b , #0:Fresh ) ) , sec ( a , #3:Fresh ) ) ) , nil ] )

 Intruder knowledge
 --------------------------
e ( exp ( #1:Gen , n ( a , #2:Fresh ) <+> n ( b , #0:Fresh ) ) , sec ( a , #3:Fresh ) ) !inI ,
 ( a ; #4:Name ; exp ( #1:Gen , n ( b , #0:Fresh ) ) ) !inI ,
 ( #4:Name ; exp ( #1:Gen , n ( b , #0:Fresh ) ) ) inI ,
 inst ( #1:Gen )

 Sequence of messages
 --------------------------
- ( a ) ,
 - ( #4:Name ; exp ( #1:Gen , n ( b , #0:Fresh ) ) ) ,
 + ( a ; #4:Name ; exp ( #1:Gen , n ( b , #0:Fresh ) ) ) ,
 - ( a ; #4:Name ; exp ( #1:Gen , n ( b , #0:Fresh ) ) ) ,
 + ( e ( exp ( #1:Gen , n ( a , #2:Fresh ) <+> n ( b , #0:Fresh ) ) , sec ( a , #3:Fresh ) ) ) ,
 - ( e ( exp ( #1:Gen , n ( a , #2:Fresh ) <+> n ( b , #0:Fresh ) ) , sec ( a , #3:Fresh ) ) )
```

Graphical view of Strands for each state



Strands Visualization node <(1[2]).6.4>

Window

Caption

Honest strand

Intruder strand

pk(b,n Intruder's knowledge

Message introduced by the attack

Temporal mark, the messages above it belong to the past and the ones below it to the future

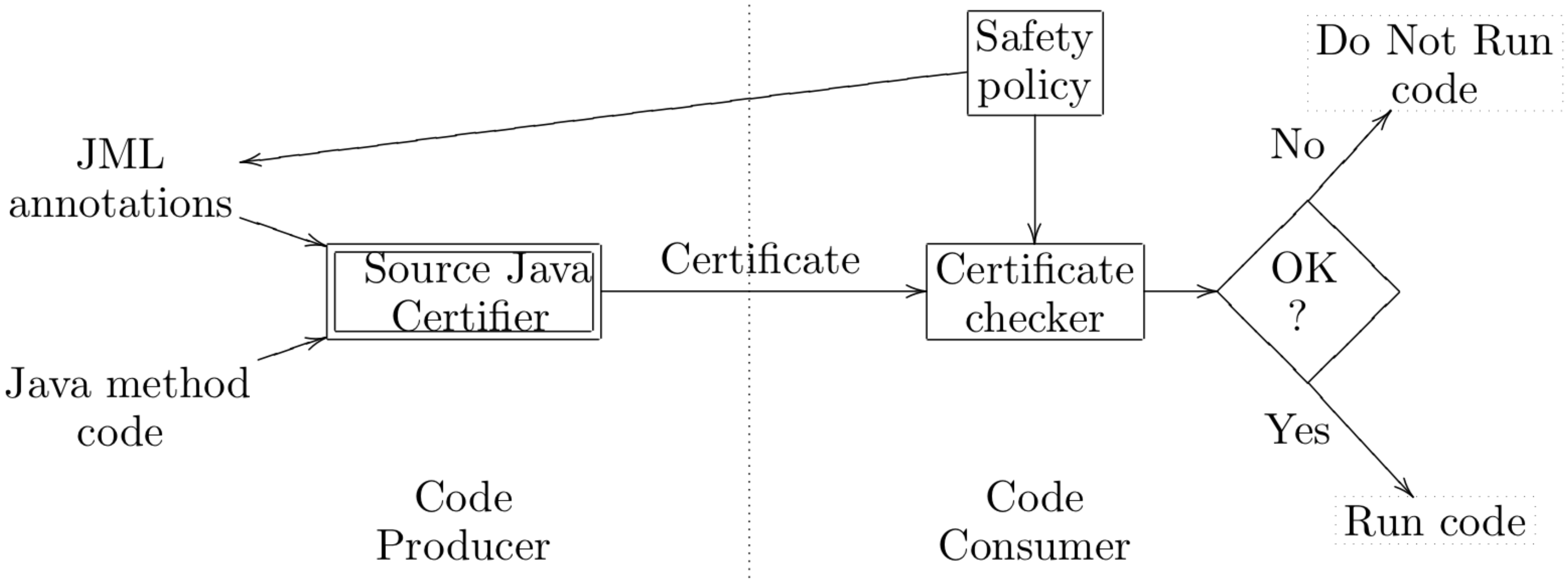
pk(b)
a
#4:Name;exp(#1:Gen,n(b,#0:Fresh))
a;b;exp(#1:Gen,n(a,#2:Fresh))
a;#4:Name;exp(g,n(a,#2:Fresh))
a;#4:Name;exp(#1:Gen,n(b,#0:Fresh))
a;b;exp(g,n(b,#0:Fresh))
e(exp(#1:Gen,n(a,#2:Fresh)
<+>n(b,#0:Fresh))
,sec(a,#3:Fresh))

A Tool for Automated Certification of Java Source Code in Maude

Mauricio Alba-Castro (Universidad Autonoma de Manizales, Colombia)
María Alpuente (Universidad Politécnica de Valencia, Spain)
Santiago Escobar (Universidad Politécnica de Valencia, Spain)
Pedro Ojeda (Universidad Politécnica de Valencia, Spain)
Daniel Romero (Universidad Nacional de Río Cuarto, Argentina)

Proof-Carrying Code Scheme in Maude

- Proof-Carrying Code based on rewriting logic (Maude)

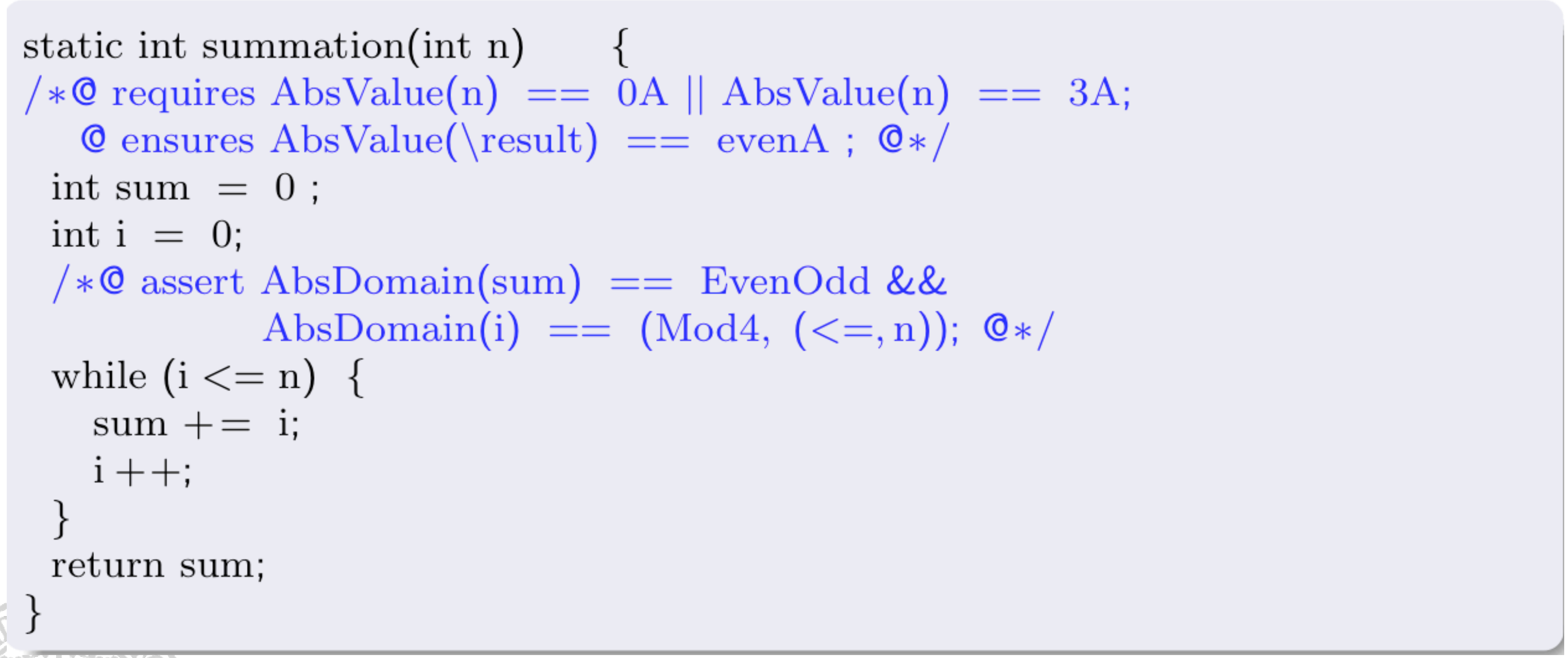


Abstract Reachability Analysis

- Producer (certificate generator):
  - Uses abstract operational semantics of Java written in Maude
  - Chooses an abstract domain suitable for the safety properties to be certified
  - Initial and final abstract reachability states are inferred from safety properties
  - Applies abstract reachability analysis
- Safety Certificate:
  - Consists of abstract rewriting sequences (performed by Maude)
  - Generated from search command in Maude
  - Demonstrate unreachability of unsafe states, i.e. those undesired states which are inferred from safety property

Safety Policies specified in JML

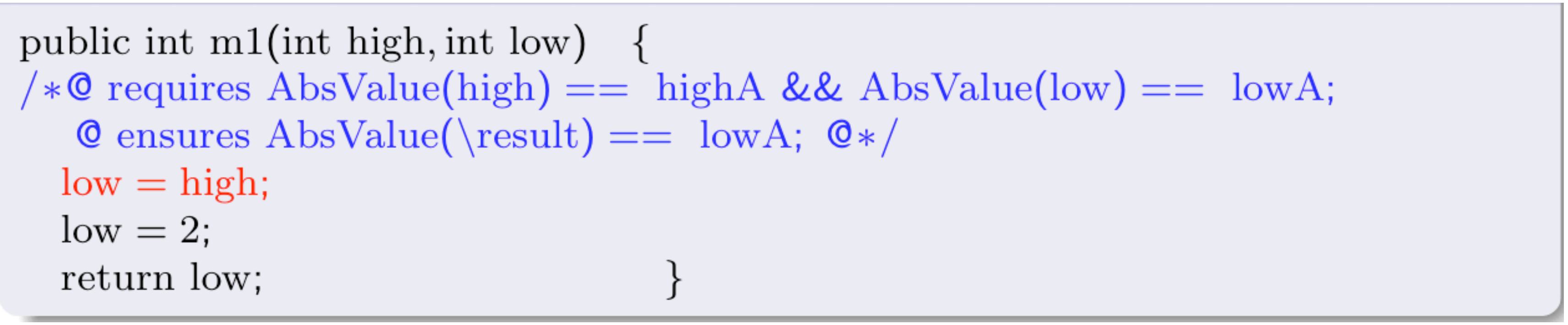
- Arithmetic:



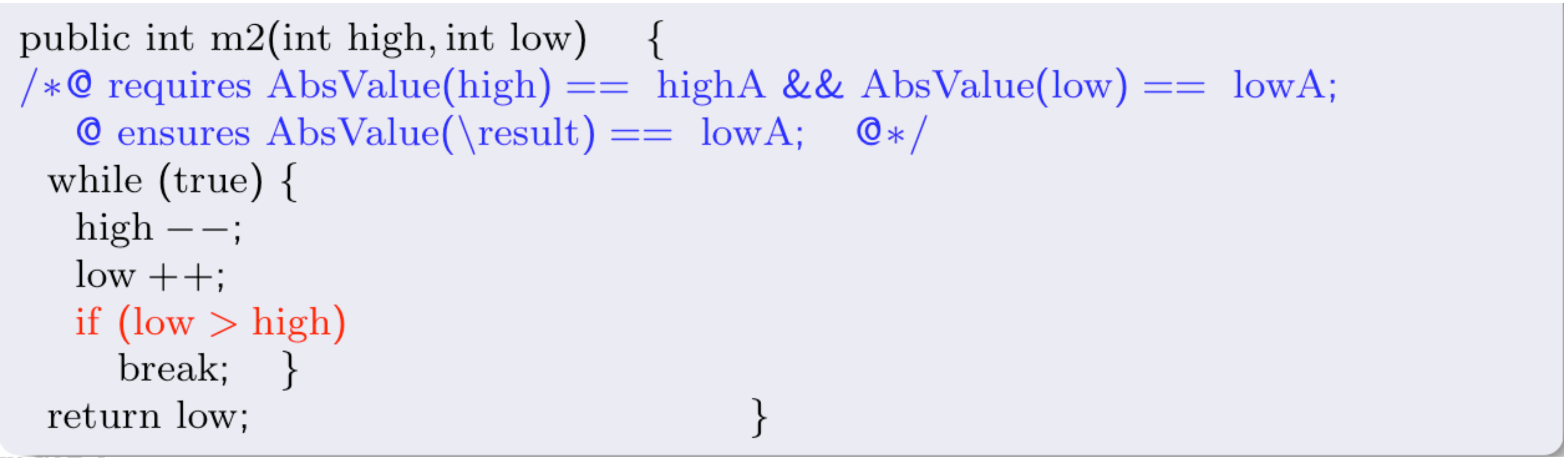
```
static int summation(int n)      {
/*@ requires AbsValue(n) == 0A || AbsValue(n) == 3A;
   @ ensures AbsValue(\result) == evenA ; @*/
  int sum = 0 ;
  int i = 0;
  /*@ assert AbsDomain(sum) == EvenOdd &&
              AbsDomain(i) == (Mod4, (<=, n)); @*/
  while (i <= n) {
     sum += i;
     i ++;
  }
  return sum;
}
```

Safety Policies specified in JML

- Non-interference:

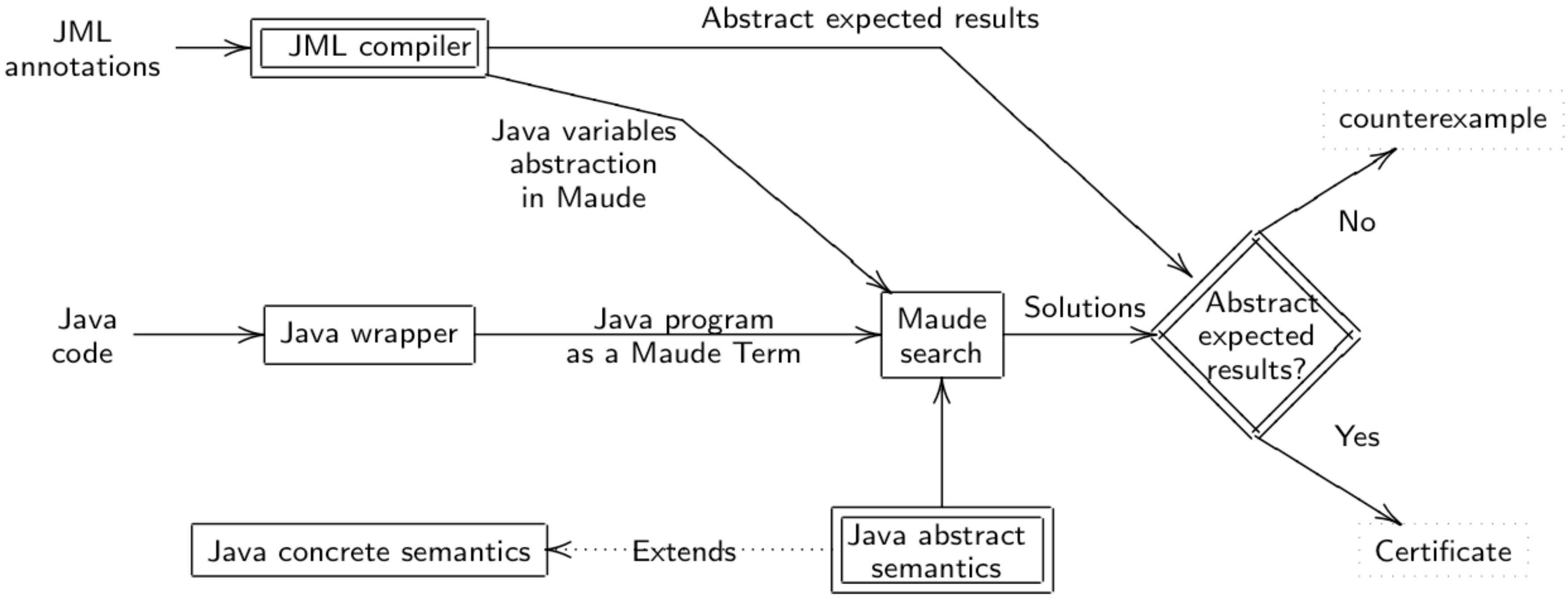


```
public int m1(int high, int low)   {
/*@ requires AbsValue(high) ==  highA && AbsValue(low) ==  lowA;
    @ ensures AbsValue(\result) ==  lowA;  @*/
   low = high;
   low = 2;
   return low;                          }
```

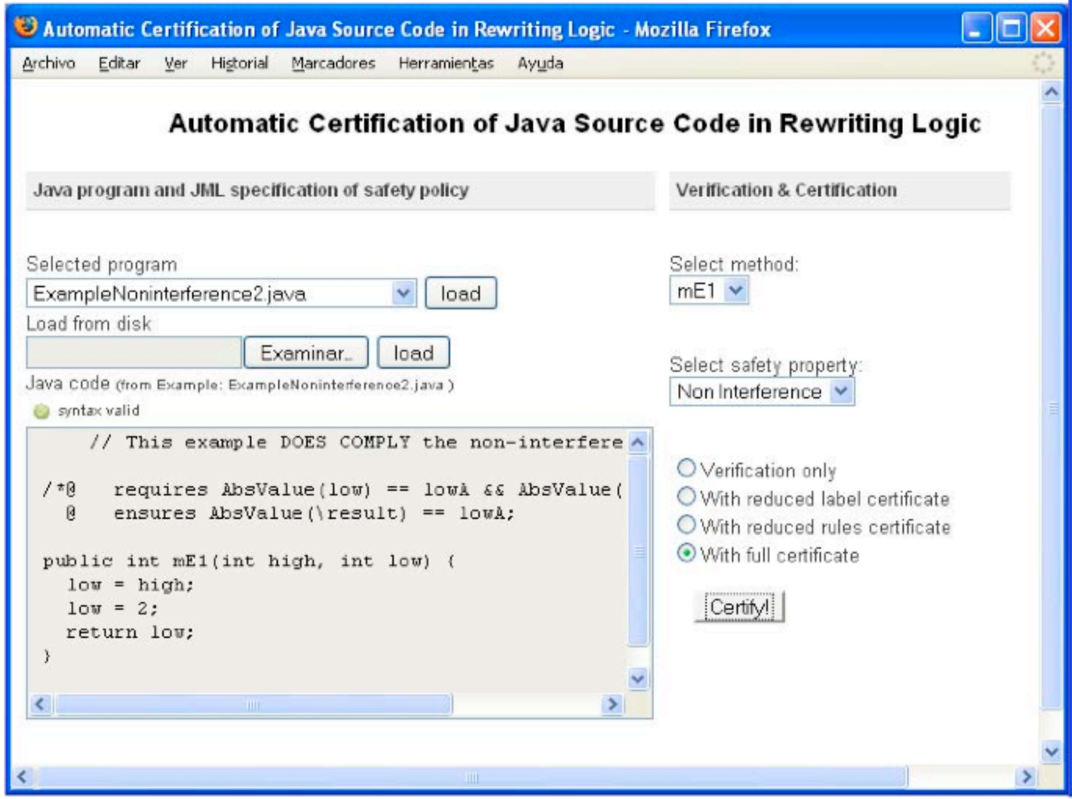


```
public int m2(int high, int low)    {
/*@ requires AbsValue(high) ==  highA && AbsValue(low) ==  lowA;
    @ ensures AbsValue(\result) ==  lowA;    @*/
  while (true) {
    high −−;
    low ++;
    if (low > high)
       break;    }
  return low;                               }
```

PCC Overview



Main Windows


Automatic Certification of Java Source Code in Rewriting Logic - Mozilla Firefox
Archivo Editar Ver Historial Marcadores Herramientas Ayuda

Automatic Certification of Java Source Code in Rewriting Logic

Java program and JML specification of safety policy

Selected program: ExampleNoninterference2.java

load
Load from disk
Examinar...
load

Java code (from Example: ExampleNoninterference2.java)

✓ syntax valid

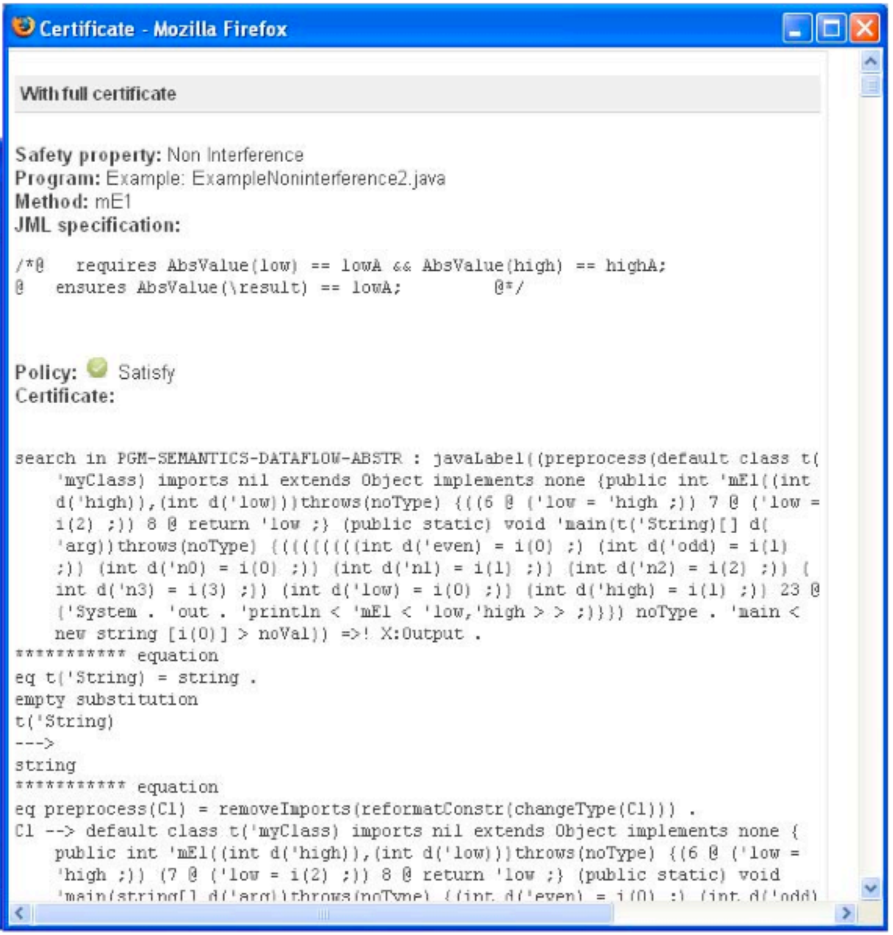
```
    // This example DOES COMPLY the non-interfere

/*@   requires AbsValue(low) == lowA && AbsValue(
  @   ensures AbsValue(\result) == lowA;

public int mE1(int high, int low) {
  low = high;
  low = 2;
  return low;
}
```

Verification & Certification

Select method: mE1

Select safety property: Non Interference

○ Verification only
○ With reduced label certificate
○ With reduced rules certificate
● With full certificate



With full certificate

Safety property: Non Interference
Program: Example: ExampleNoninterference2.java
Method: mE1
JML specification:

```
/*@   requires AbsValue(low) == lowA && AbsValue(high) == highA;
@   ensures AbsValue(\result) == lowA;          @*/
```

Policy: ✓ Satisfy
Certificate:

```
search in PGM-SEMANTICS-DATAFLOW-ABSTR : javaLabel((preprocess(default class t(
    'myClass) imports nil extends Object implements none {public int 'mE1((int
    d('high)),(int d('low)))throws(noType) {((6 @ ('low = 'high ;)) 7 @ ('low =
    i(2) ;)) 8 @ return 'low ;} (public static) void 'main(t('String)[] d(
    'arg))throws(noType) {((((((((((int d('even) = i(0) ;) (int d('odd) = i(1)
    ;)) (int d('n0) = i(0) ;)) (int d('n1) = i(1) ;)) (int d('n2) = i(2) ;)) (
    int d('n3) = i(3) ;)) (int d('low) = i(0) ;)) (int d('high) = i(1) ;)) 23 @
    ('System . 'out . 'println < 'mE1 < 'low,'high > > ;)}}}) noType . 'main <
    new string [i(0)] > noVal)) =>! X:Output .
*********** equation
eq t('String) = string .
empty substitution
t('String)
--->
string
*********** equation
eq preprocess(C1) = removeImports(reformatConstr(changeType(C1))) .
C1 --> default class t('myClass) imports nil extends Object implements none {
    public int 'mE1((int d('high)),(int d('low)))throws(noType) {(6 @ ('low =
    'high ;)) (7 @ ('low = i(2) ;)) 8 @ return 'low ;} (public static) void
    'main(string[] d('arg))throws(noType) {(int d('even) = i(0) ;) (int d('odd)
```