Behavioral Program Queries using Logic Source Code Templates

Coen De Roover[▲] Johan Brichau[✲]

[▲] Programming Technology Lab - *Vrije Universiteit Brussel* - Belgium

[✲] Département d'Ingénierie Informatique - *Université catholique de Louvain* - Belgium

Context: Program Queries

identify elements

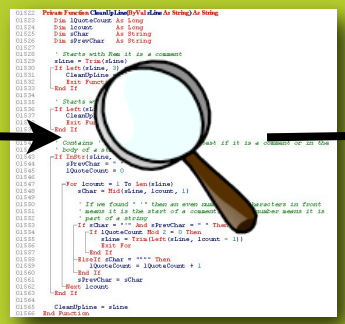
implementing

faulty
well-known
prescribed
suboptimal

behavior

effective identification
+
declarative specification

abstract definition



concrete instances

Context: Program Queries

identify elements

implementing

faulty
well-known
prescribed
suboptimal

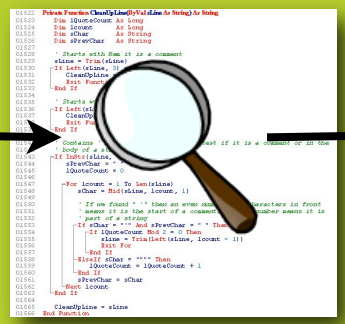
behavior

heterogeneous code

similar behavior

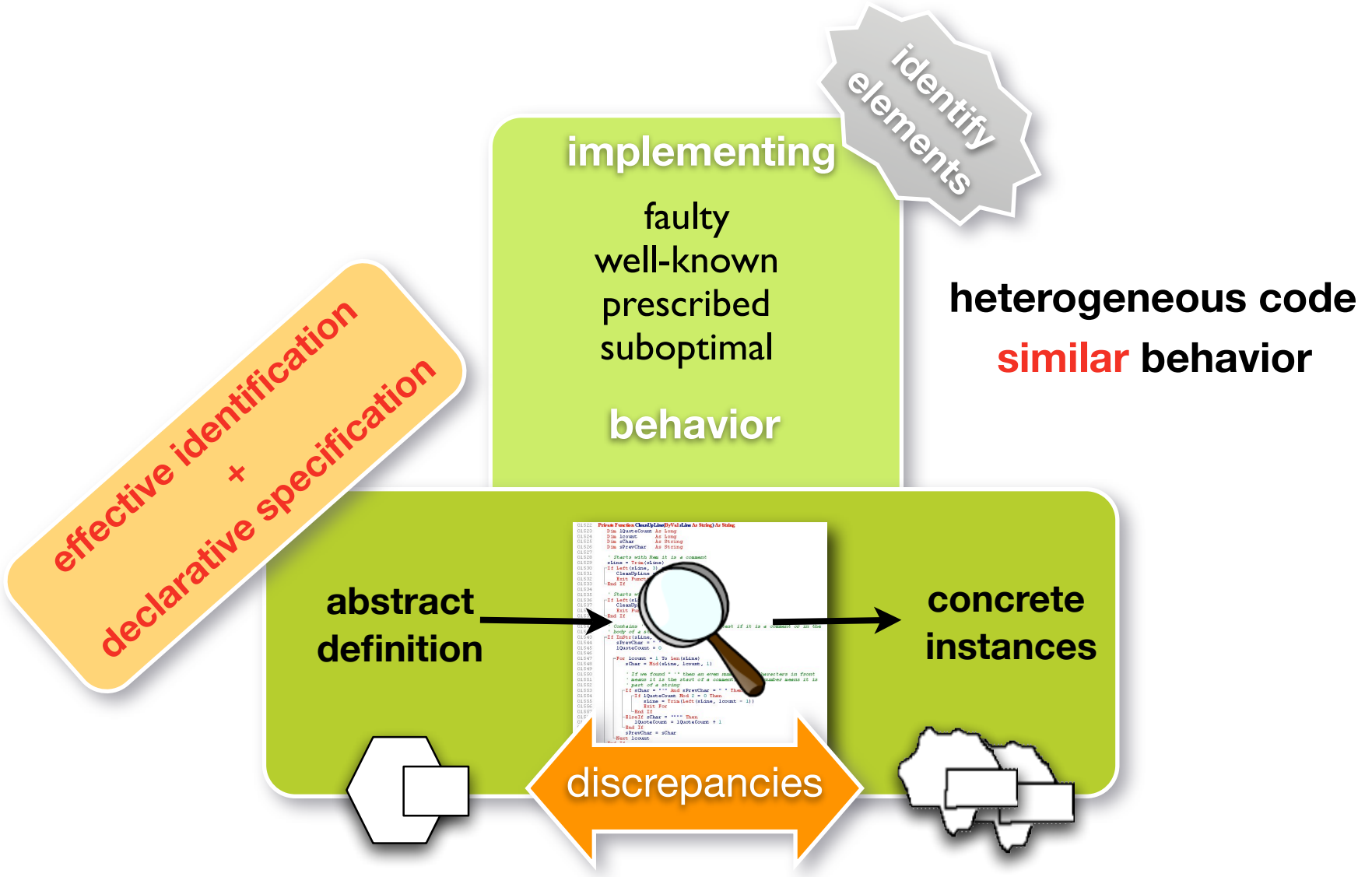
effective identification
+
declarative specification

abstract definition



concrete instances

Context: Program Queries



Motivating Example: Detecting Getter Methods

```
1 ?m getsFragment: ?g ofFieldDeclaration: ?f in: ?c if
2  ?f isFieldDeclarationInClassDeclaration: ?c,
3  ?f fieldDeclarationHasFragment: ?g,
4  ?g variableDeclarationFragmentHasName: ?name,
5  ?m isMethodDeclarationInClassDeclaration: ?c
6  ?m methodDeclarationHasBody: block(?s),
7  ?s contains: returnStatement(?name)
```

Should detect getter methods in an imaginary* LMP language.

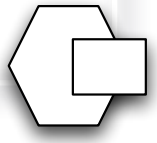
* any similarities with Soul minus the extensions I'm about to present are purely incidental and unintentional ;)

Motivating Example: Detecting Getter Methods

```
1 ?m getsFragment: ?g ofFieldDeclaration: ?f in: ?c if
2  ?f isFieldDeclarationInClassDeclaration: ?c,
3  ?f fieldDeclarationHasFragment: ?g,
4  ?g variableDeclarationFragmentHasName: ?name,
5  ?m isMethodDeclarationInClassDeclaration: ?c
6  ?m methodDeclarationHasBody: block(?s),
7  ?s contains: returnStatement(?name)
```

Should detect getter methods in an imaginary* LMP language.

```
class Y {
 private X var;
 public X getVar() {
      return var;
 }
}
```



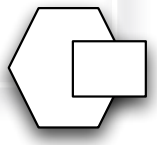
* any similarities with Soul minus the extensions I'm about to present are purely incidental and unintentional ;)

Motivating Example: Detecting Getter Methods

```
1 ?m getsFragment: ?g ofFieldDeclaration: ?f in: ?c if
2  ?f isFieldDeclarationInClassDeclaration: ?c,
3  ?f fieldDeclarationHasFragment: ?g,
4  ?g variableDeclarationFragmentHasName: ?name,
5  ?m isMethodDeclarationInClassDeclaration: ?c
6  ?m methodDeclarationHasBody: block(?s),
7  ?s contains: returnStatement(?name)
```

Should detect getter methods in an imaginary* LMP language.

```
class Y {
 private X var;
 public X getVar() {
      return var;
 }
}
```



```
class Y {
 private X var;
 private Y self() {
   return this;
 }
 public X getVar() {
      return this.self().var;
 }
}
```

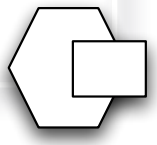
* any similarities with Soul minus the extensions I'm about to present are purely incidental and unintentional ;)

Motivating Example: Detecting Getter Methods

```
1 ?m getsFragment: ?g ofFieldDeclaration: ?f in: ?c if
2  ?f isFieldDeclarationInClassDeclaration: ?c,
3  ?f fieldDeclarationHasFragment: ?g,
4  ?g variableDeclarationFragmentHasName: ?name,
5  ?m isMethodDeclarationInClassDeclaration: ?c
6  ?m methodDeclarationHasBody: block(?s),
7  ?s contains: returnStatement(?name)
```

Should detect getter methods in an imaginary* LMP language.

```
class Y {
 private X var;
 public X getVar() {
      return var;
 }
}
```

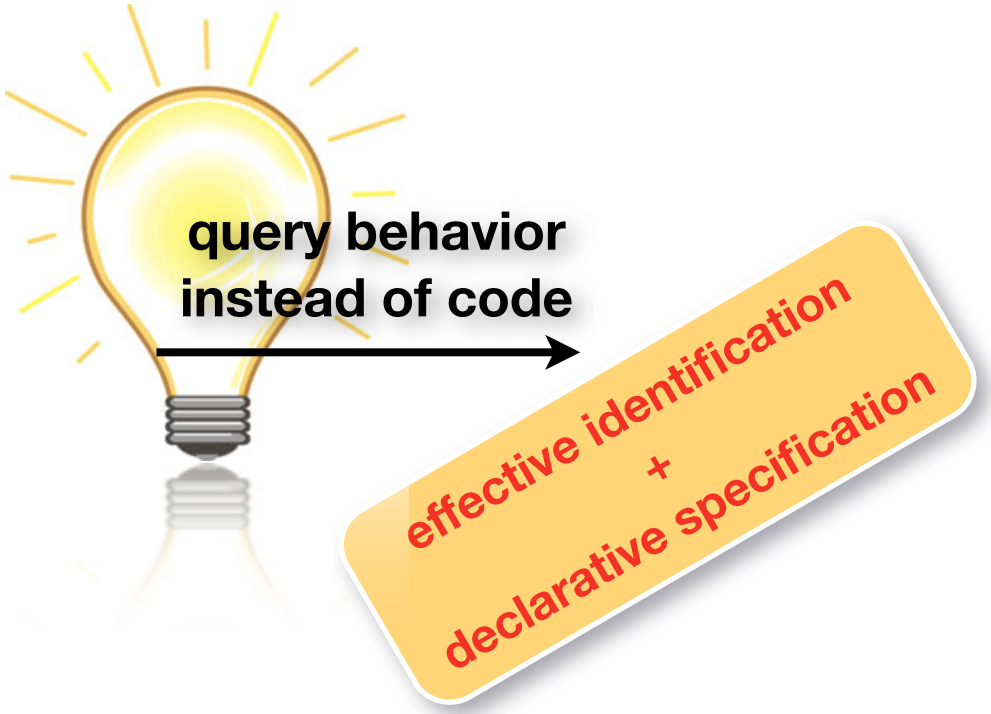


```
class Y {
 private X var;
 private Y self() {
   return this;
 }
 public X getVar() {
      return this.self().var;
 }
}
```

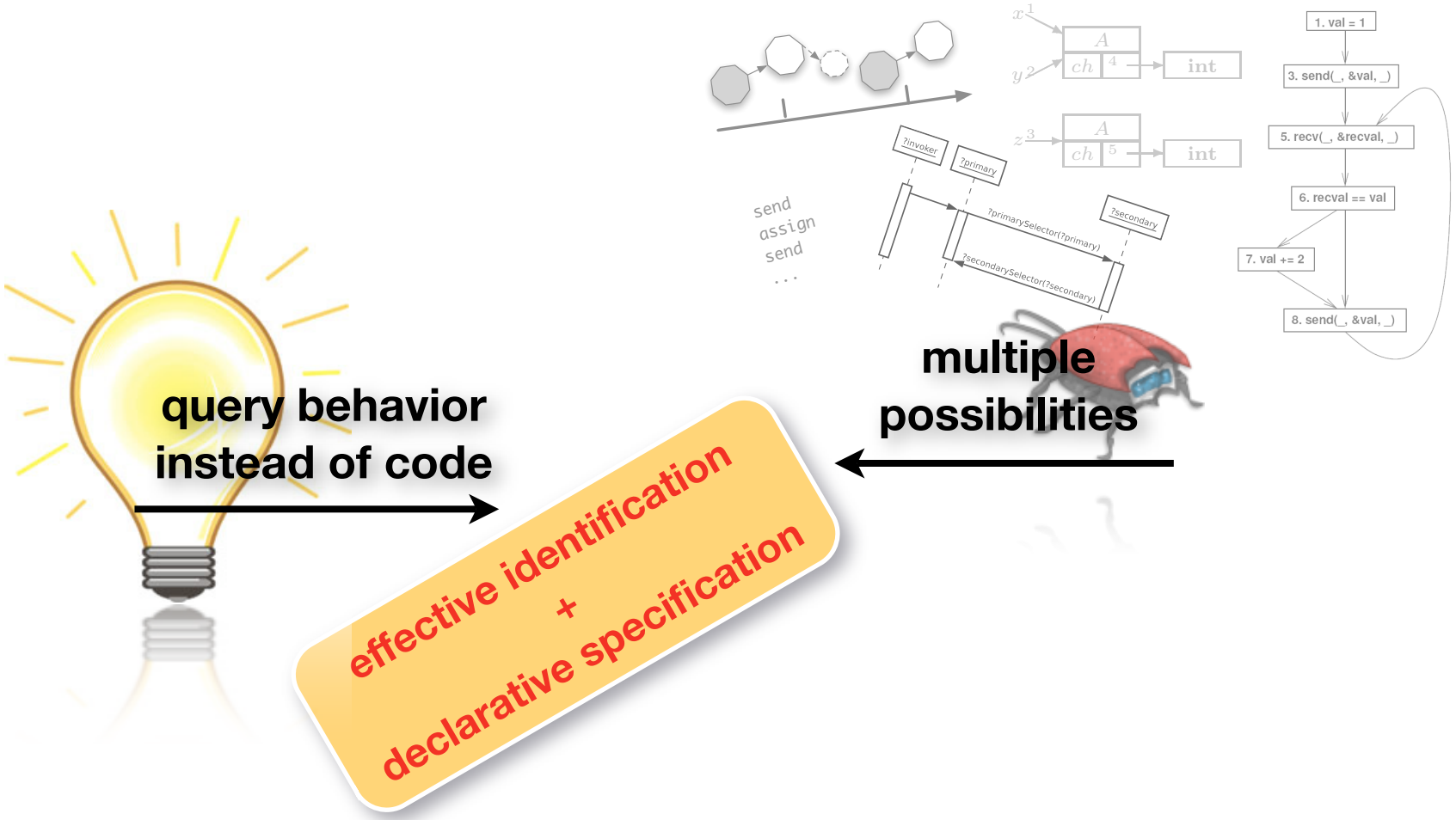
```
class Y {
 private X var;
 public X notGetVar(X var) {
      return var;
   }
}
```

* any similarities with Soul minus the extensions I'm about to present are purely incidental and unintentional ;)

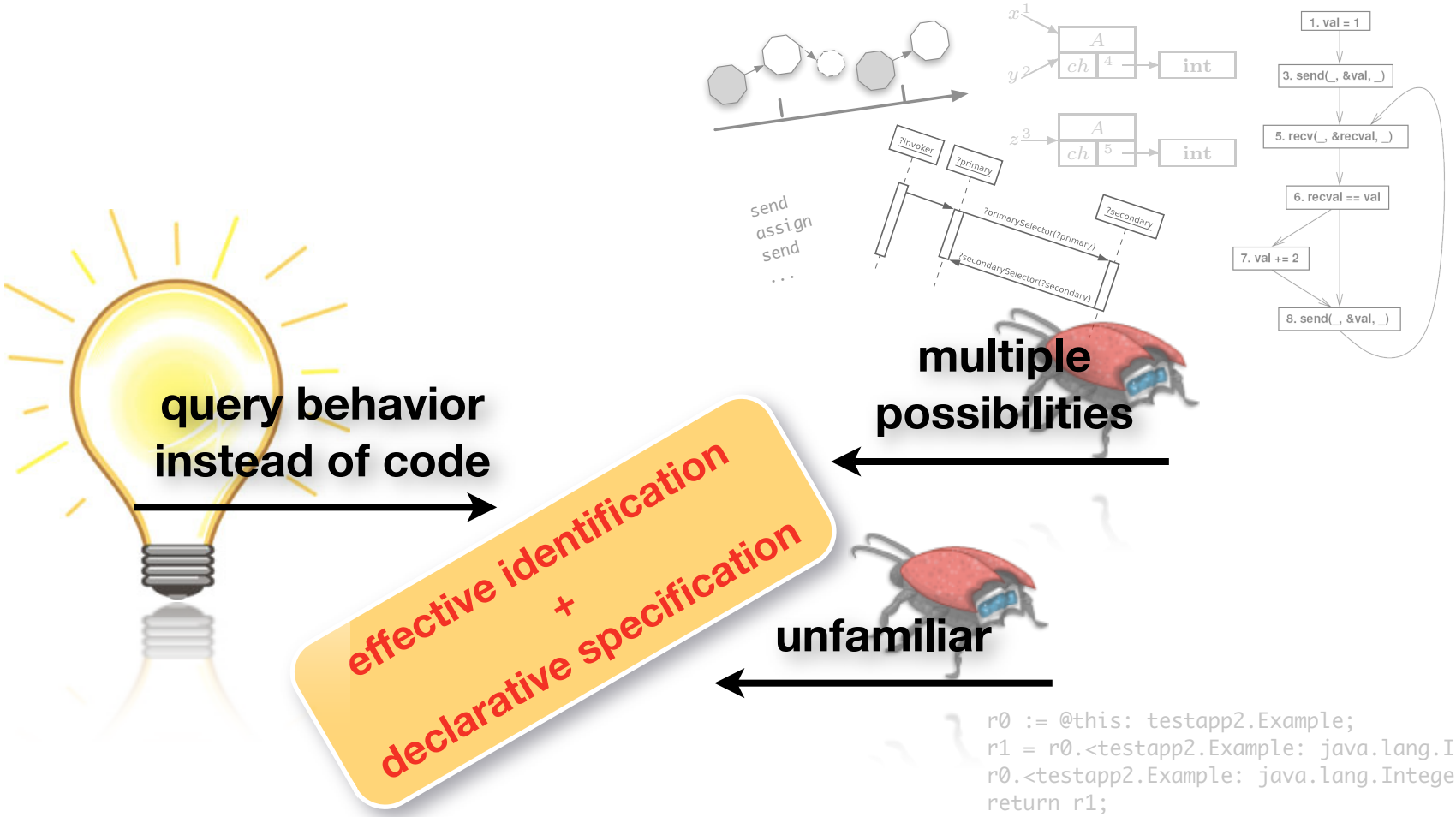
Isn't there a Straightforward Solution?



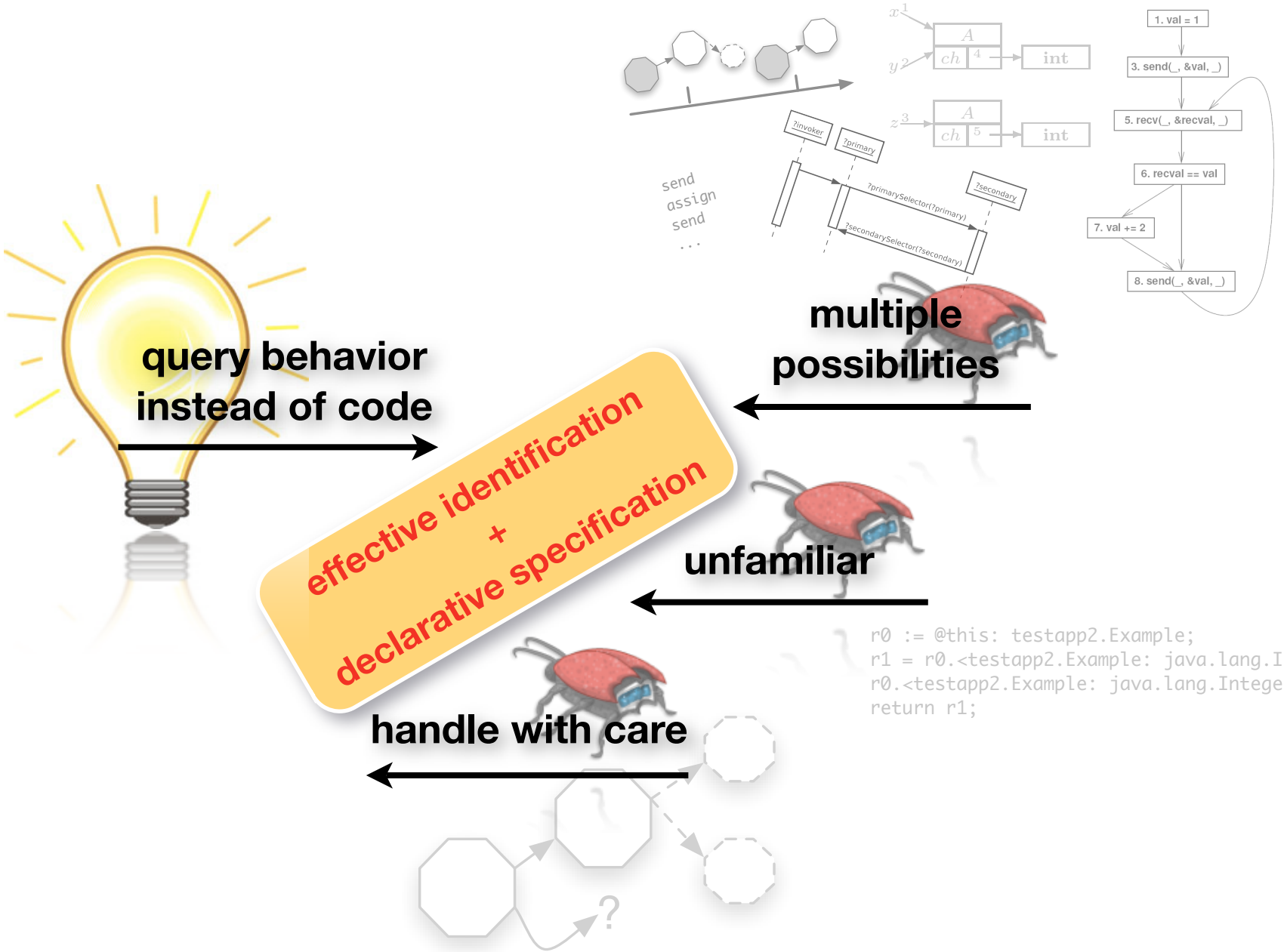
Isn't there a Straightforward Solution?



Isn't there a Straightforward Solution?



Isn't there a Straightforward Solution?



So, instead ..

```
if jtClassDeclaration(?c){
        class ?c {
            private ?type ?field;
            public ?type ?name() { return ?field; }
        }
}
```

```
public Integer gethour() {
    return this.hour;
}
public Integer gethourlazy() {
    if(hour==null)
      hour = this.currentHour();
    return hour;
}
public Integer getBuffer() {
    Integer temp;
    temp = this.self().buffer;
    return temp;
}
```

Solution Cornerstones

declarative pattern specification

concrete source code templates

integrated in logic language

effective pattern identification

resolved by fuzzy logic program

behavioral similarity determined statically

by open unification framework

Solution Cornerstones

declarative pattern specification

user familiarity

concrete source code templates **integrated in logic language**

effective pattern identification

resolved by fuzzy logic program

behavioral similarity determined statically

by open unification framework

Solution Cornerstones

declarative pattern specification

user familiarity → **concrete source code templates**

template composition → **integrated in logic language**

effective pattern identification

resolved by fuzzy logic program

behavioral similarity determined statically

by open unification framework

Solution Cornerstones

declarative pattern specification

user familiarity

concrete source code templates

template composition

integrated in logic language

uniform **unification** across templates and ordinary conditions

effective pattern identification

resolved by fuzzy logic program

behavioral similarity determined statically by open unification framework

Solution Cornerstones

declarative pattern specification

user familiarity → **concrete source code templates**

template composition → **integrated in logic language**

uniform **unification** across templates and ordinary conditions

effective pattern identification

similarity template/match ← **resolved by fuzzy logic program**

behavioral similarity determined statically by open unification framework

Solution Cornerstones

declarative pattern specification

user familiarity → **concrete source code templates**

template composition → **integrated in logic language**

uniform **unification** across templates and ordinary conditions

effective pattern identification

similarity template/match ← **resolved by fuzzy logic program** → customizable

behavioral similarity determined statically by open unification framework

Solution Cornerstones

declarative pattern specification

user familiarity → **concrete source code templates**

template composition → **integrated in logic language**

uniform **unification** across templates and ordinary conditions

effective pattern identification

similarity template/match

resolved by fuzzy logic program → customizable

behavioral similarity determined statically by open unification framework

transparant to user

Solution Cornerstones

declarative pattern specification

user familiarity → **concrete source code templates**

template composition → **integrated in logic language**

uniform **unification** across templates and ordinary conditions

effective pattern identification

similarity template/match

resolved by fuzzy logic program → customizable

behavioral similarity determined statically by open unification framework

family of analyses

transparant to user

Solution Cornerstones

declarative pattern specification

user familiarity → **concrete source code templates**

template composition → **integrated in logic language**

uniform **unification** across templates and ordinary conditions

effective pattern identification

similarity template/match

resolved by fuzzy logic program → customizable

behavioral similarity determined statically

by open unification framework → approximated results

family of analyses

transparant to user

Fuzzy Logic Programming

logic of quantified truth

rules annotated with partial **truth degrees**

✓ ***confidence*** *in instances detected by rule*

fuzzy logic

weighted logic rules

$q : c$ **if** $q_1,...,q_n$ **where** $c \in]0,1]$

similar to f-Prolog [1990:liu]

Fuzzy Logic Programming

logic of quantified truth

rules annotated with partial **truth degrees**

✓ ***confidence*** *in instances detected by rule*

fuzzy logic

weighted logic rules

$q : c$ **if** $q_1,...,q_n$ **where** $c \in]0,1]$

confidence in conclusion q given absolute truth of $q_1,...,q_n$

similar to f-Prolog [1990:liu]

Fuzzy Logic Programming

logic of quantified truth

rules annotated with partial **truth degrees**

✓ ***confidence*** *in instances detected by rule*

fuzzy logic

fuzzy resolution procedure

conclusions from **partially satisfied premises**

✓ *detect partially adhering instances*

confidence in conclusion `q` given absolute truth of $q_1,...,q_n$

weighted logic rules

$q : c$ **if** $q_1,...,q_n$ **where** $c \in]0,1]$

fuzzy resolution procedure

$T(q) = c * \min(T(q1),...,T(qn))$

similar to f-Prolog [1990:liu]

Fuzzy Logic Programming

logic of quantified truth

rules annotated with partial **truth degrees**

✓ ***confidence*** *in instances detected by rule*

fuzzy logic

fuzzy resolution procedure

conclusions from **partially satisfied premises**

✓ *detect partially adhering instances*

confidence in conclusion q given absolute truth of $q_1,\ldots,q_n$

weighted logic rules

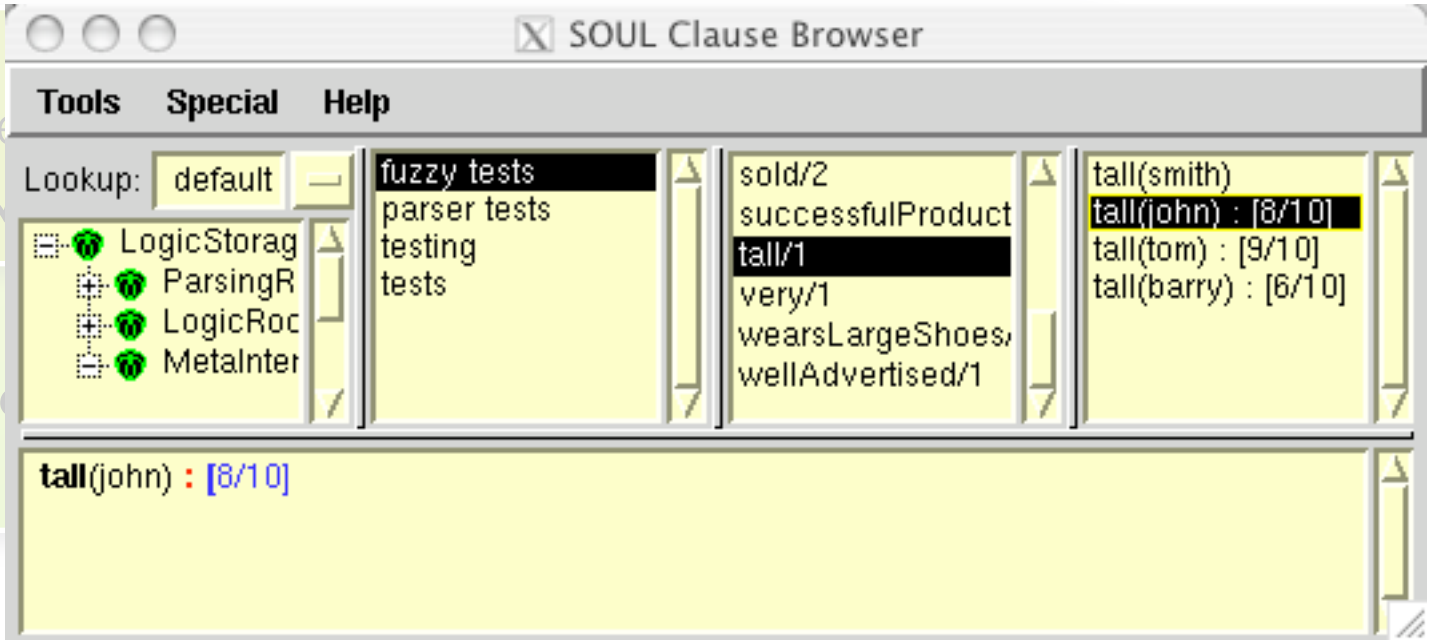
$q : c$ **if** $q_1,\ldots,q_n$ **where** $c \in]0,1]$

fuzzy resolution procedure

$\tau(q) = c * \min(\tau(q1),\ldots,\tau(qn))$

similar to f-Prolog [1990:liu]

Fuzzy Logic Programming



confidence in conclusion q given absolute truth of $q_1,...,q_n$

weighted logic rules

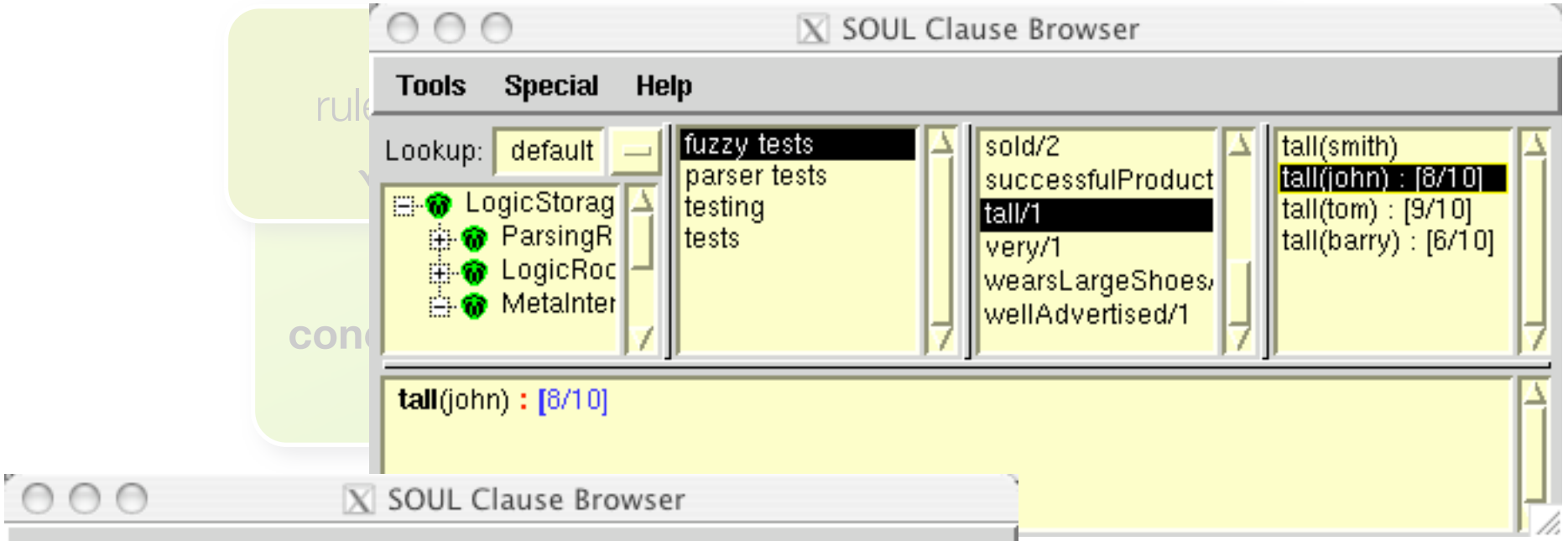
$q : c$ **if** $q_1,...,q_n$ where $c \in]0,1]$

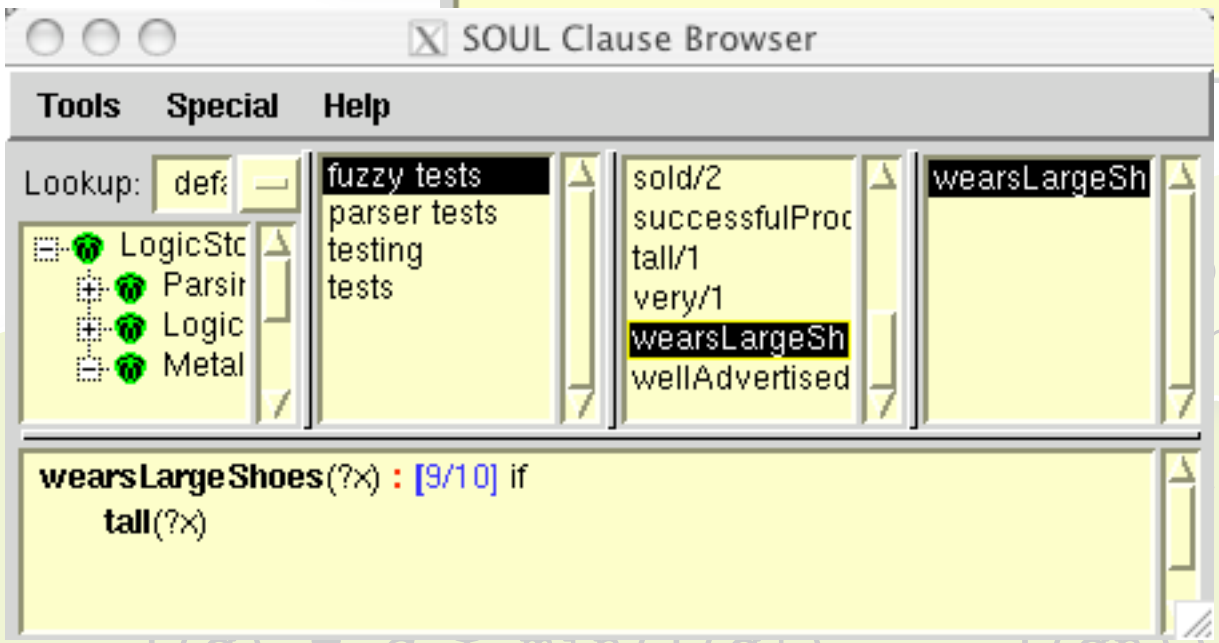
fuzzy resolution procedure

$T(q) = c * \min(T(q1),...,T(qn))$

similar to f-Prolog [1990:liu]

Fuzzy Logic Programming

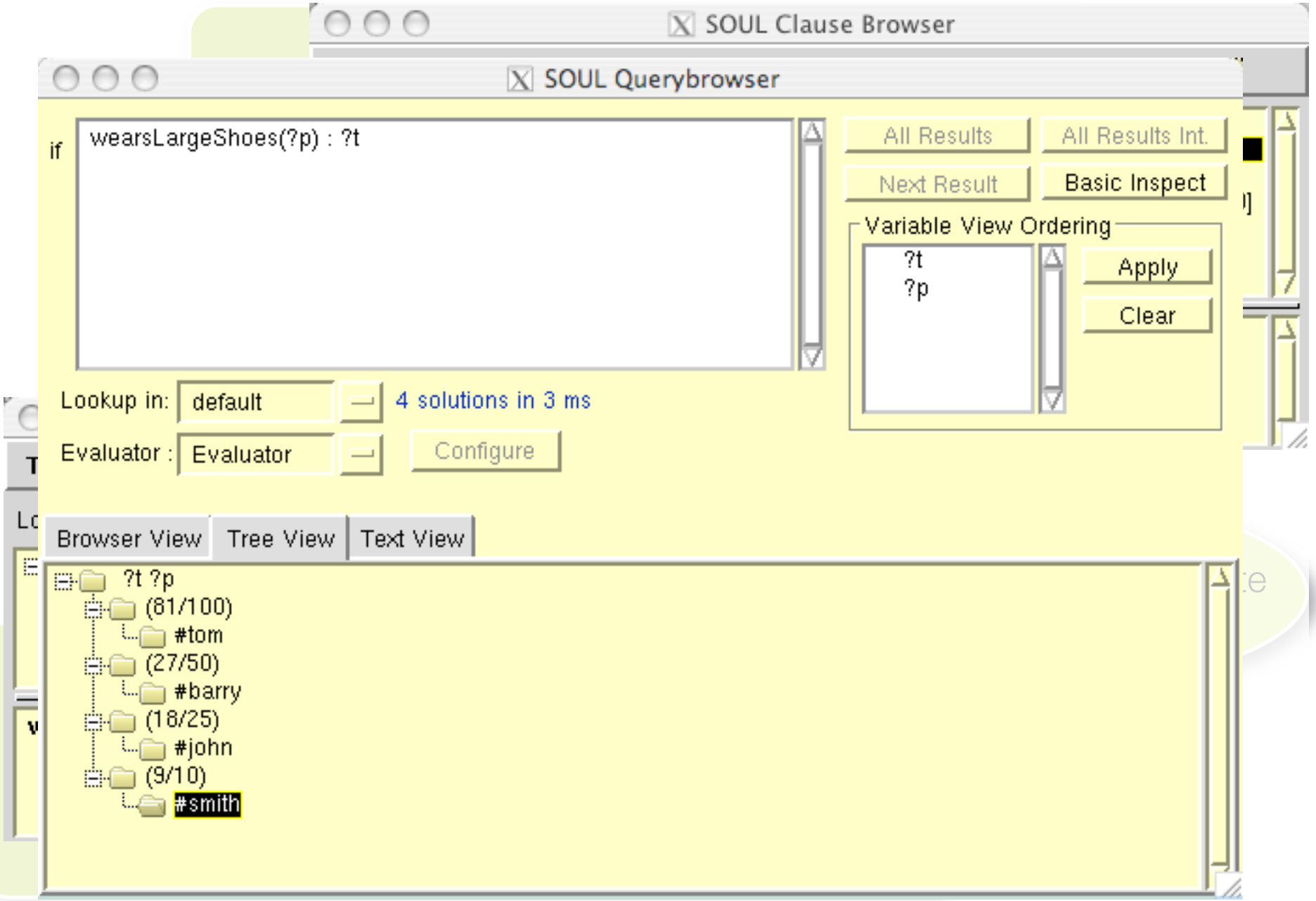




confidence
nclusion q given absolute
ruth of $q_1, \ldots, q_n$

similar to
f-Prolog
[1990:liu]

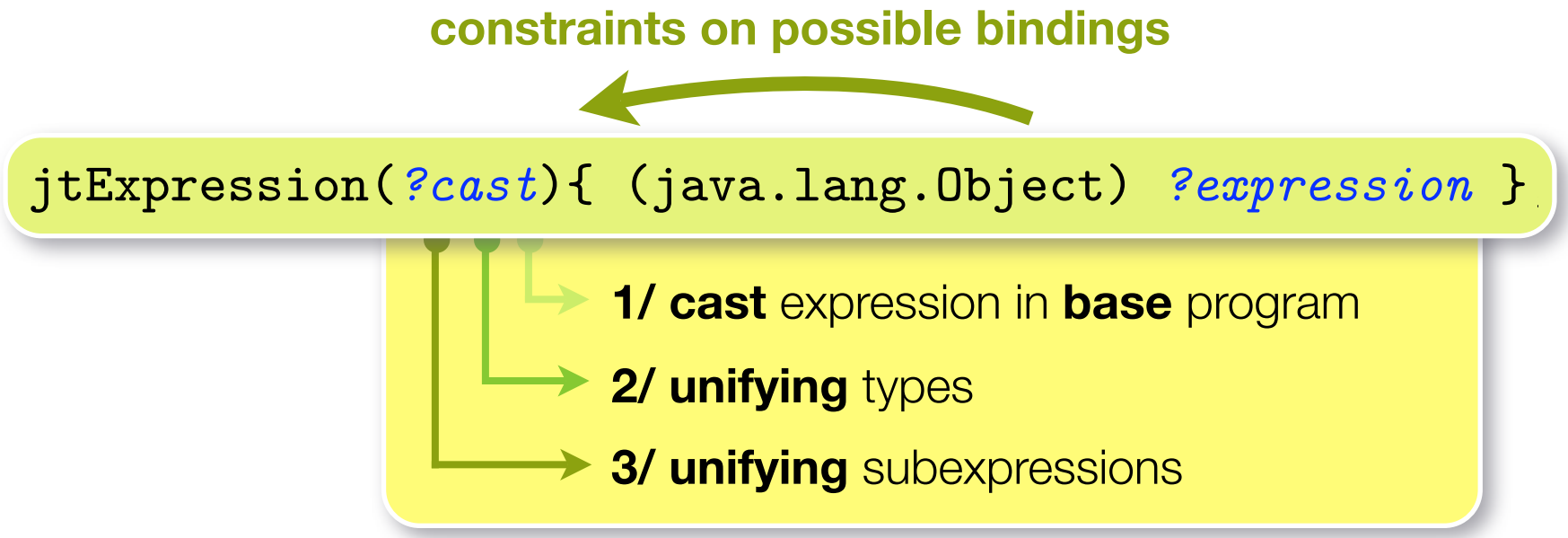
Fuzzy Logic Programming



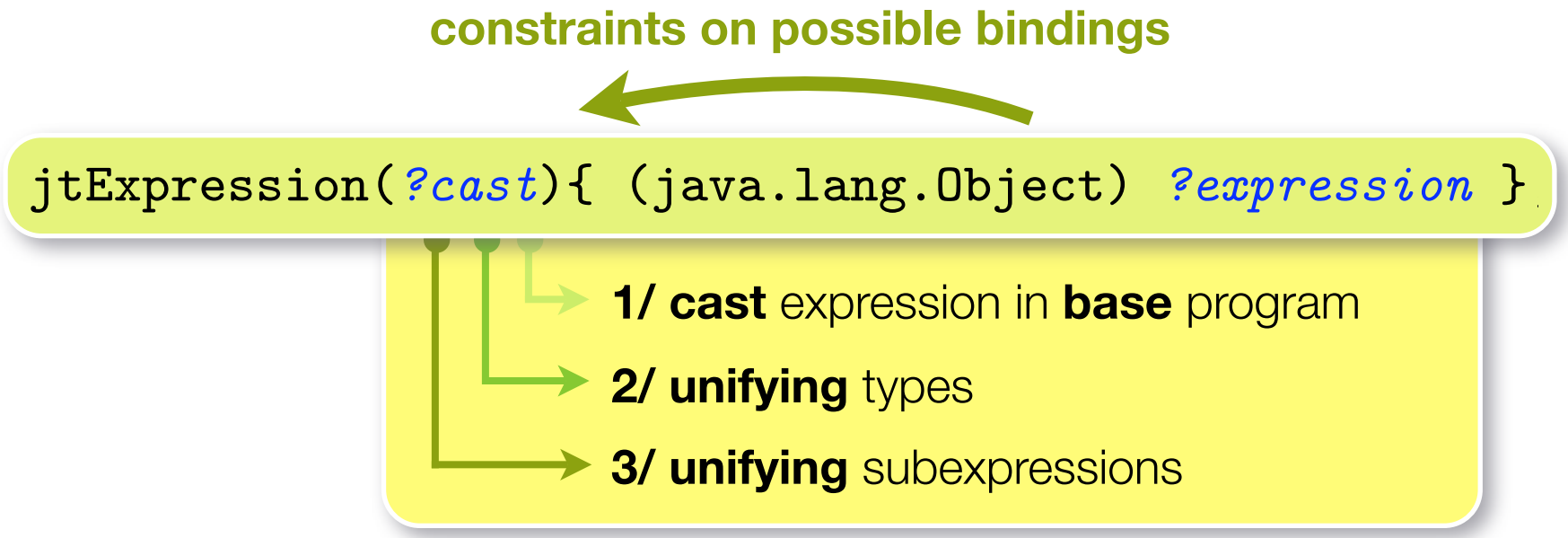
Open Template Compilation

```
jtExpression(?cast){ (java.lang.Object) ?expression }
```

Open Template Compilation



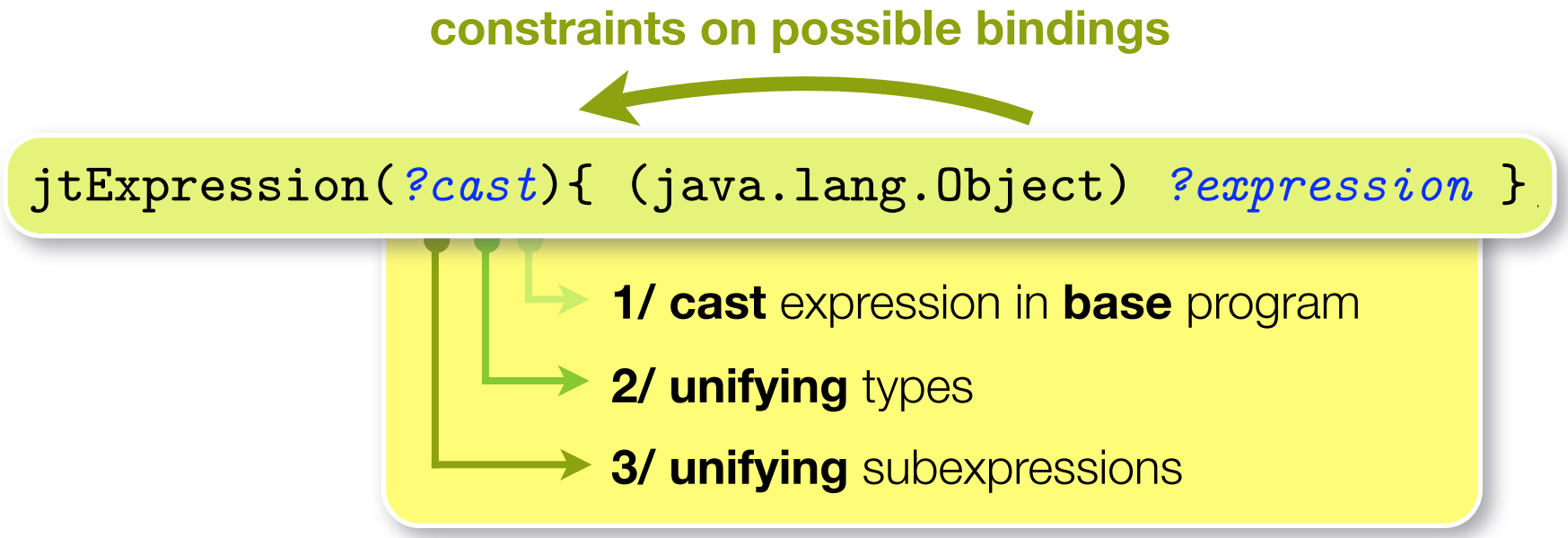
Open Template Compilation



compiled by fuzzy logic program called by interpreter

results in multiple logic queries for each parse tree of a template

Open Template Compilation



compiled by fuzzy logic program called by interpreter

results in multiple logic queries for each parse tree of a template

weighted by their **tolerance** for **mismatches**

```
if jtStatement(?block){ ?s1; ?s2; ?s3;}
```

layers of indirection, intertwining statement, loops, ...

Multiple Program Representations

Multiple Program Representations

Abstract Syntax Tree

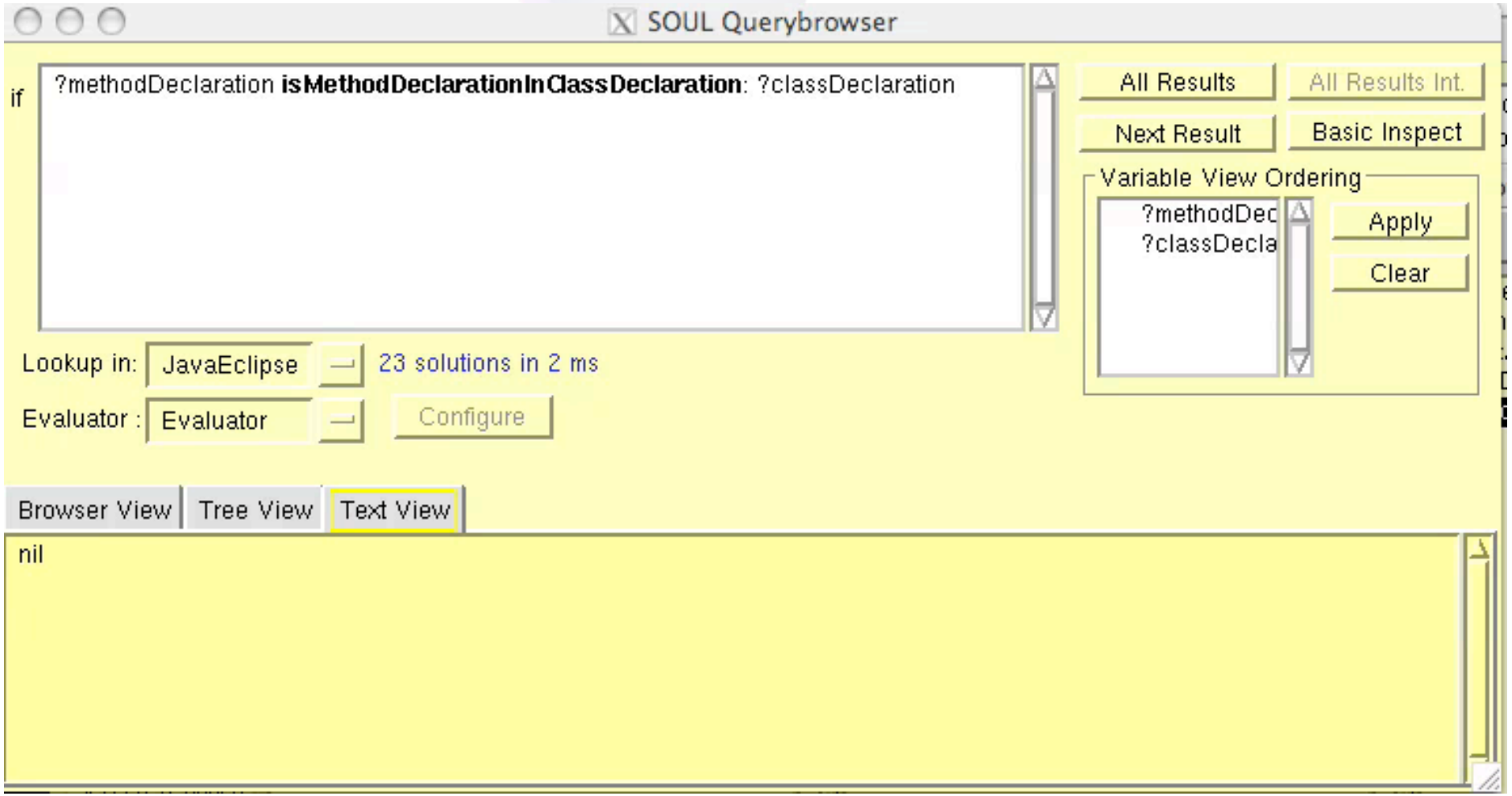
live  `org.eclipse.jdt.core.dom.ASTNode` instances

NOT reified as logic facts

Multiple Program Representations

Abstract Syntax Tree

live  `org.eclipse.jdt.core.dom.ASTNode` instances



Multiple Program Representations

Abstract Syntax Tree

live  `org.eclipse.jdt.core.dom.ASTNode` instances

NOT reified as logic facts

Multiple Program Representations

Abstract Syntax Tree

live  `org.eclipse.jdt.core.dom.ASTNode` instances

NOT reified as logic facts

Dataflow Analysis: points-to

safe approximation of run-time objects a reference points to

x.y →

```
{ AllocNode 3 new java.lang.Integer in method
<testapp2.Example: void main(java.lang.String[])>,
...
AllocNode 7 new java.lang.Integer in method
<testapp2.Example: void main(java.lang.String[])> }
```

Multiple Program Representations

Abstract Syntax Tree

live  `org.eclipse.jdt.core.dom.ASTNode` instances

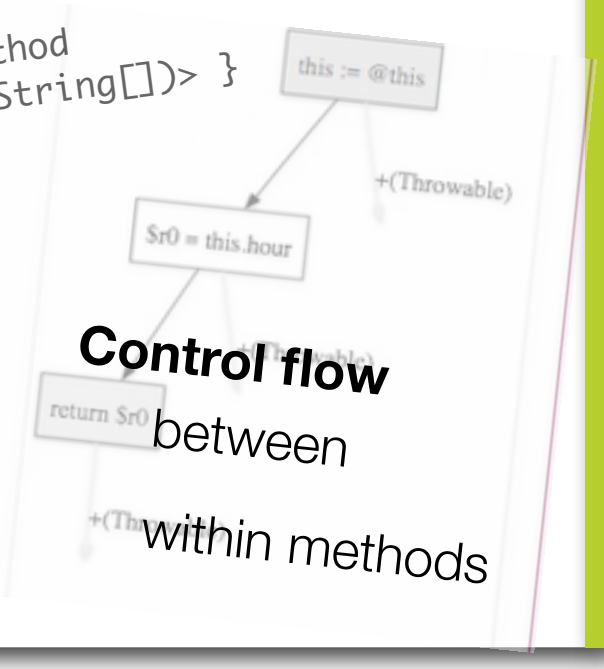
NOT reified as logic facts

Dataflow Analysis: points-to

safe approximation of run-time objects a reference points to

x.y →

```
{ AllocNode 3 new java.lang.Integer in method
<testapp2.Example: void main(java.lang.String[])>,
...
AllocNode 7 new java.lang.Integer in method
<testapp2.Example: void main(java.lang.String[])> }
```

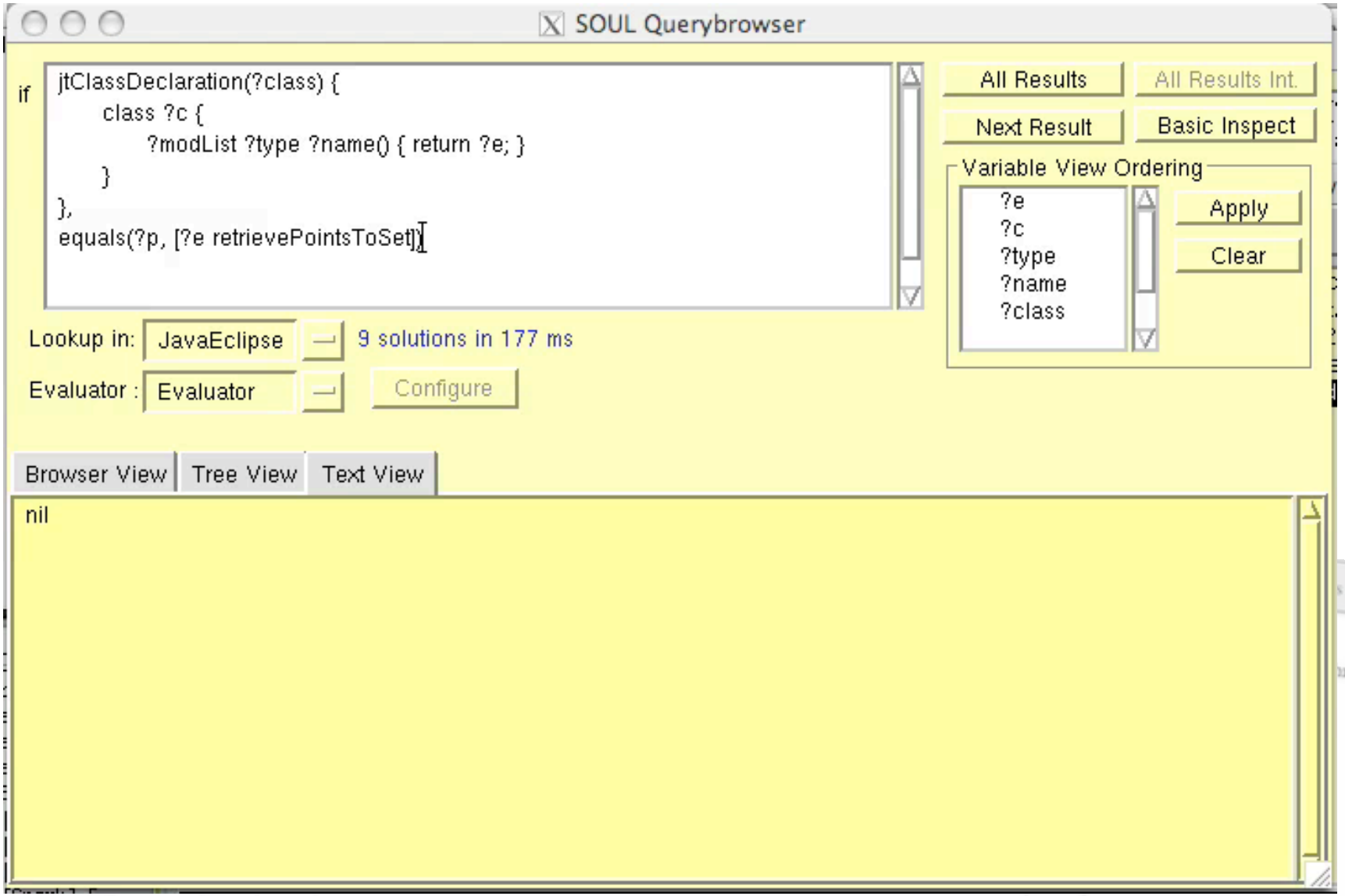


Control flow

between

within methods

Multiple Program Representations



within methods

Multiple Program Representations

Abstract Syntax Tree

live  `org.eclipse.jdt.core.dom.ASTNode` instances

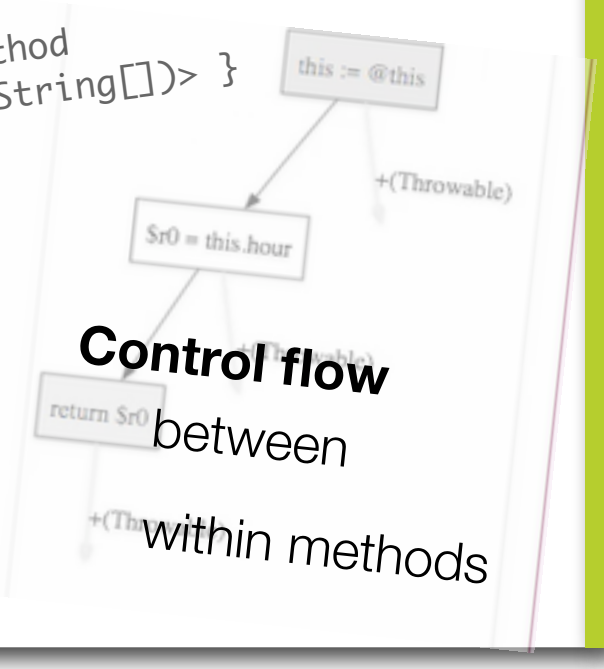
NOT reified as logic facts

Dataflow Analysis: points-to

safe approximation of run-time objects a reference points to

x.y →

```
{ AllocNode 3 new java.lang.Integer in method
<testapp2.Example: void main(java.lang.String[])>,
...
AllocNode 7 new java.lang.Integer in method
<testapp2.Example: void main(java.lang.String[])> }
```



Control flow

between

within methods

Smalltalk **Open Unification** **L**anguage

Unification with logic terms

Unification with other  **instances**

Smalltalk **Open Unification** Language

Unification with logic terms

easily identify elements of interest

```
1 if ?c isCompilationUnit,
2    [?c types size > 1]

3 if compilationUnit(packageDeclaration(simpleName(['testapp'])),?,?) isCompilationUnit

4 if ?c isCompilationUnit,
5    ?c hasPackage: ?p,
6    ?p hasName: ?n,
7    ?n isSimpleName,
8    ?n hasIdentifier: ['testapp']
```

Unification with other  instances

Smalltalk **Open Unification** Language

Unification with logic terms powered by structural reflection

easily identify elements of interest

```
1 if ?c isCompilationUnit,
2    [?c types size > 1]

3 if compilationUnit(packageDeclaration(simpleName(['testapp'])),?,?) isCompilationUnit

4 if ?c isCompilationUnit,
5    ?c hasPackage: ?p,
6    ?p hasName: ?n,
7    ?n isSimpleName,
8    ?n hasIdentifier: ['testapp']
```

Unification with other  **instances**

Smalltalk **Open Unification** Language

Unification with logic terms powered by structural reflection

easily identify elements of interest

```
1 if ?c isCompilationUnit,
2    [?c types size > 1]

3 if compilationUnit(packageDeclaration(simpleName(['testapp'])),?,?) isCompilationUnit

4 if ?c isCompilationUnit,
5    ?c hasPackage: ?p,
6    ?p hasName: ?n,
7    ?n isSimpleName,
8    ?n hasIdentifier: ['testapp']
```

Unification with other  **instances** overrides default identity comparison

implement recurring comparisons

```
Expression ≟ ParenthesizedExpression

Type ≟ TypeDeclaration

SimpleName ≟ QualifiedName
```

Smalltalk **Open Unification** Language

Unification with logic terms powered by structural reflection

easily identify elements of interest

```
1 if ?c isCompilationUnit,
2    [?c types size > 1]

3 if compilationUnit(packageDeclaration(simpleName(['testapp'])),?,?) isCompilationUnit

4 if ?c isCompilationUnit,
5    ?c hasPackage: ?p,
6    ?p hasName: ?n,
7    ?n isSimpleName,
8    ?n hasIdentifier: ['testapp']
```

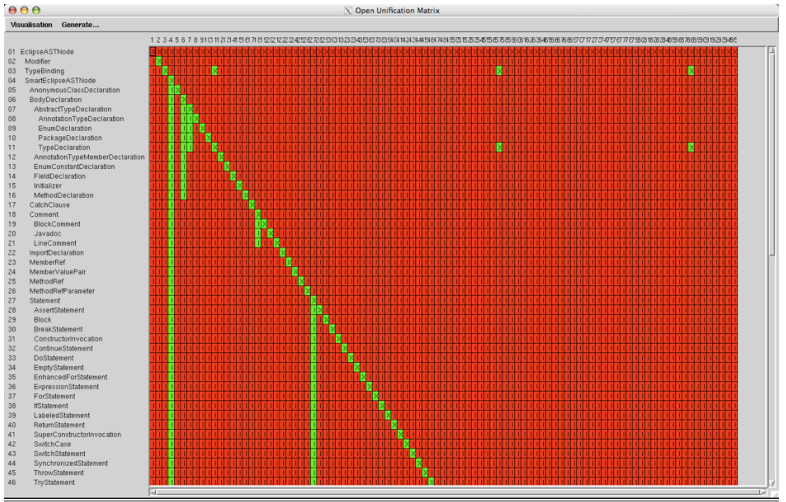
Unification with other  **instances** overrides default identity comparison

implement recurring comparisons

Expression $\stackrel{?}{=}$ ParenthesizedExpression

Type... $\stackrel{?}{=}$ TypeDeclaration

SimpleName $\stackrel{?}{=}$ QualifiedName



Smalltalk **Open Unification** Language

Unification with logic terms powered by structural reflection

easily identify elements of interest

```
1 if ?c isCompilationUnit,
2    [?c types size > 1]

3 if compilationUnit(packageDeclaration(simpleName(['testapp'])),?,?) isCompilationUnit

4 if ?c isCompilationUnit,
5    ?c hasPackage: ?p,
6    ?p hasName: ?n,
7    ?n isSimpleName,
8    ?n hasIdentifier: ['testapp']
```

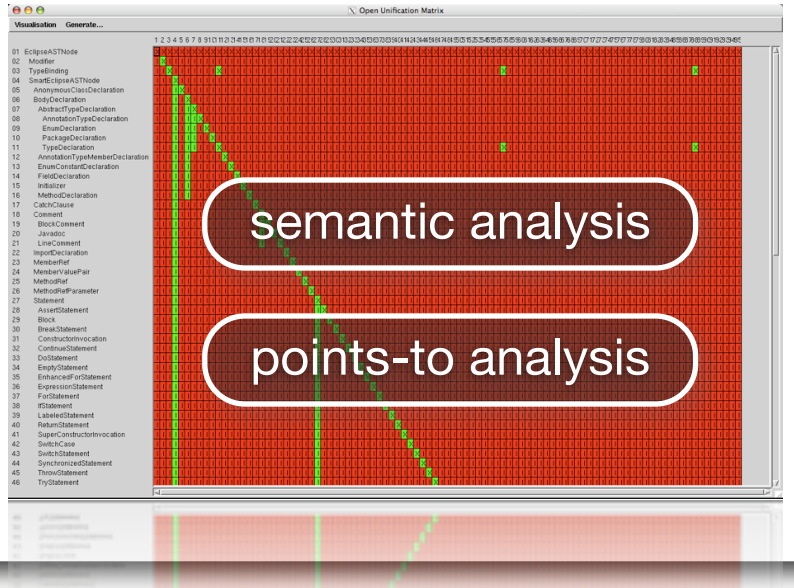
Unification with other  **instances** overrides default identity comparison

implement recurring comparisons

Expression $\stackrel{?}{=}$ ParenthesizedExpression

Type... $\stackrel{?}{=}$ TypeDeclaration

SimpleName $\stackrel{?}{=}$ QualifiedName



Unification based on Semantic Analysis

across logic and template conditions

Unification based on Semantic Analysis

Conciseness

across logic and template conditions

Unification based on Semantic Analysis

Conciseness

Correctness

across logic and template conditions

Unification based on Semantic Analysis

name resolution

Conciseness

```
1 if jtClassDeclaration(?class){
2      class FooClass extends testapp2.Example implements Collection {}
3 }
4 if jtClassDeclaration(?class){
5      class FooClass extends Example implements java.util.Collection {}
6 }
```

```
package testapp2;
public class FooClass extends testapp2.Example implements Collection {}
```

Correctness

across logic and template conditions

Unification based on Semantic Analysis

name resolution

Conciseness

```
1 if jtClassDeclaration(?class){
2      class FooClass extends testapp2.Example implements Collection {}
3 }
4 if jtClassDeclaration(?class){
5      class FooClass extends Example implements java.util.Collection {}
6 }
```

```
package testapp2;
public class FooClass extends testapp2.Example implements Collection {}
```

scoping rules

Correctness

```
1 ?m getsFragment: ?g ofFieldDeclaration: ?f in: ?c if
2  ?f isFieldDeclarationInClassDeclaration: ?c,
3  ?f fieldDeclarationHasFragment: ?g,
4  ?g variableDeclarationFragmentHasName: ?name,
5  ?m isMethodDeclarationInClassDeclaration: ?c
6  ?m methodDeclarationHasBody: block(?s),
7  ?s contains: returnStatement(?name)
```

```
private Integer f;
public Integer notGettingF(Integer f) { return f; }
```

across logic and template conditions

Unification based on Points-to Analysis

```
if jtStatement(?s1){ return ?expression; },
   jtStatement(?s2){ return ?expression; },
   differs(?s1, ?s2)
```

Unification based on Points-to Analysis

```
if jtStatement(?s1){ return ?expression; },
   jtStatement(?s2){ return ?expression; },
   differs(?s1, ?s2)
```

?s1 = return foo;	?s2
	return foo;
	return this.foo;
	return this.self().foo;
	x = foo; return x;
	return o.returnArgument(foo);

syntactically differing return statements possibly returning overlapping sets of objects

Unification based on Points-to Analysis

```
if jtStatement(?s1){ return ?expression; },
   jtStatement(?s2){ return ?expression; },
   differs(?s1, ?s2)
```

constraint over values returned at run-time

Behavioral Similarity

?s1 = return foo;	?s2
	return foo;
	return this.foo;
	return this.self().foo;
	x = foo; return x;
	return o.returnArgument(foo);

syntactically differing return statements
possibly returning overlapping sets of objects

Template Resolution Semantics

design guideline

close to concrete source code of prototypical implementation

match many implementation variants

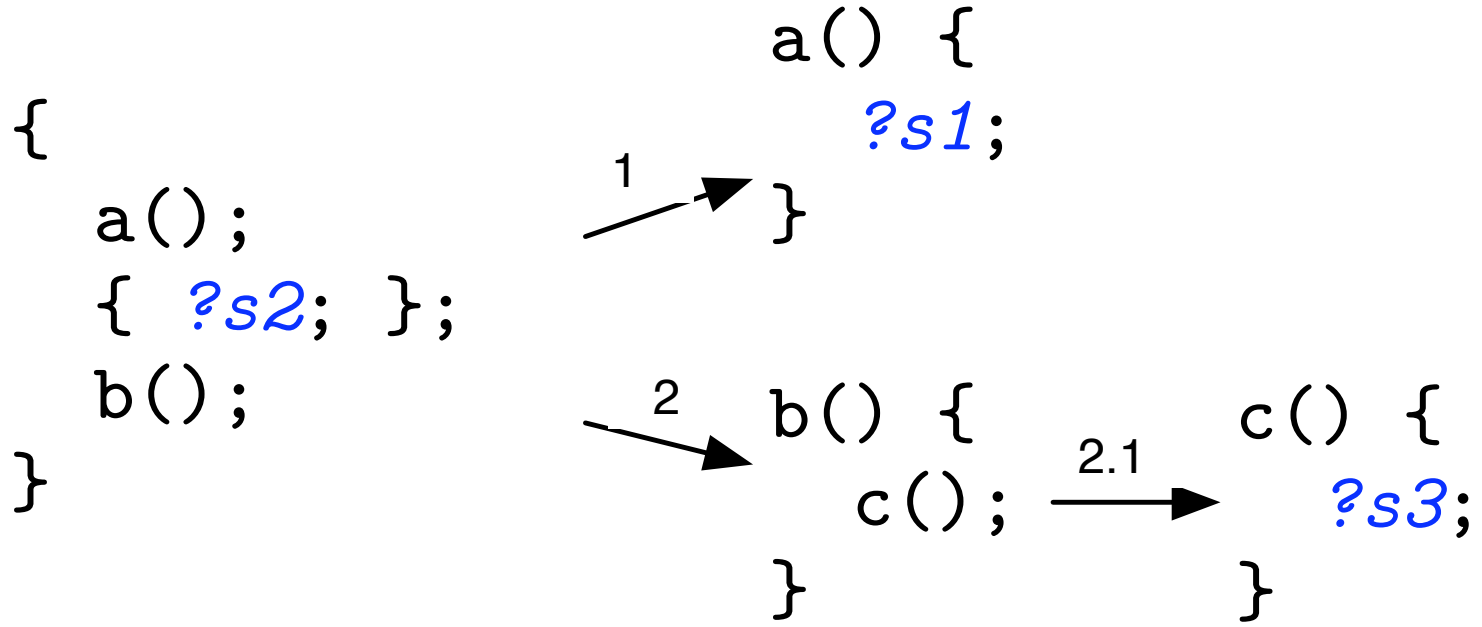
interpreted as freely as possible

BUT every single element in template adds constraints

```
while(?cond) ?s;    vs    while(?cond) { ?s; }
```

HOWEVER: an open implementation, so amenable by application programmer

Template Semantics: Sequences




```
if jtStatement(?block){ ?s1; ?s2; ?s3;}
```

1/ control flow constraint

?s's in interprocedural control-flow of *?block*

?s2 follows: *?s1*, *?s3* follows: *?s2*

2/ kind constraint (statement or expression statement)

Template Semantics: Statements and Expressions

```
public A method() {
   A local = ?x.m();
   return local;
}
```

```
jtMethodDeclaration(?m) {
   ?modList ?type ?name(?argList) {
         return ?x.m();
   }
}
```

- has a **return statement** `return(?val)` among its **lexical** statements
- has an expression `?e` which matches `?x.m()`
  - in its inter-procedural **control flow** (before the statement statement)
- the statement's returned expression `?val` **unifies with** `?e`

An Example: Concurrent Modification Exceptions

```
if jtStatement(?s) {
  while(?iterator.hasNext()) {
      ?collection.add(?element);
  }
},
jtExpression(?iterator){?collection.iterator()}
```

```
public List list;

public void insertElement(Object x) {
 Iterator i = list.iterator();
 while(i.hasNext()) {
   Object o = i.next();
   operation(x, (Collection) this.self().list);
 }
}

public void operation(Object o, Collection c) {
  c.add(o);
}
```

An Example: Concurrent Modification Exceptions

```
if jtStatement(?s) {
  while(?iterator.hasNext()) {
      ?collection.add(?element);
  }
},
jtExpression(?iterator){?collection.iterator()}
```

```
public List list;

public void insertElement(Object x) {
 Iterator i = list.iterator();
 while(i.hasNext()) {
   Object o = i.next();
   operation(x, (Collection) this.self().list);
 }
}

public void operation(Object o, Collection c) {
  c.add(o);
}
```

?s

in cflow of `while`

unify according to dataflow

Summary

concrete code templates integrated in logic program query language

- unification uniform across regular and template conditions
- composition by logic connectives
- open template compilation rules

quantified similarity template-match

- fuzzy logic

open unification framework

- incorporates static analyses in user-transparant way
  - query conciseness & correctness
  - statically determined behavioral similarity

The End