Vrije Universiteit Brussel

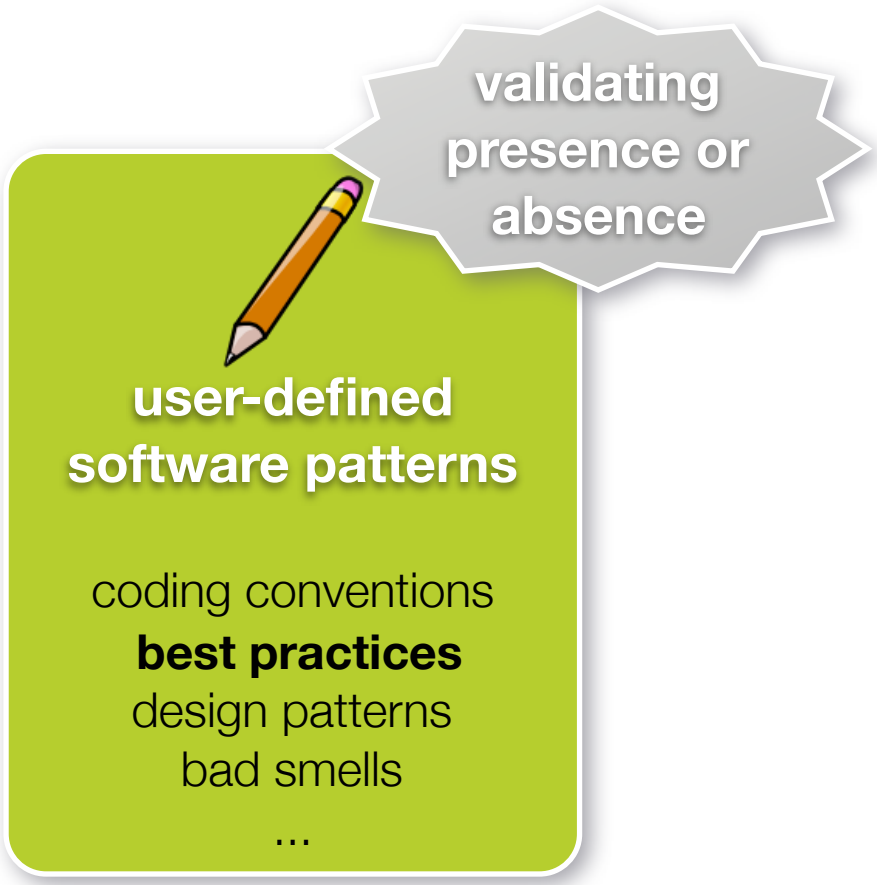
Combining **Fuzzy Logic** and **Behavioral Similarity** for Non-Strict Program Validation

Coen De Roover[▲], Johan Brichau[▽], Theo D'Hondt[▲]

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

▽ Laboratoire d'Informatique Fondamentale de Lille - *University of Lille* - France

Software Quality Assurance



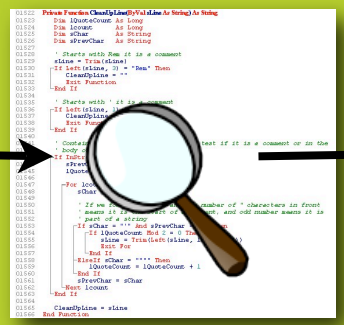
Software Quality Assurance

validating presence or absence

user-defined software patterns

coding conventions
best practices
design patterns
bad smells
...

pattern detection tool

abstract pattern definition →  → **concrete pattern instances**

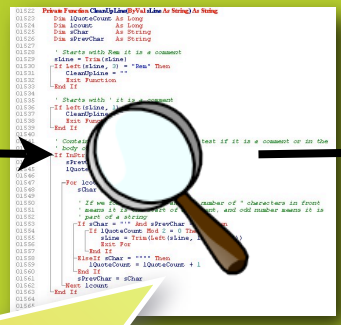
Software Quality Assurance

validating presence or absence

pattern detection tool

user-defined software patterns

abstract pattern definition

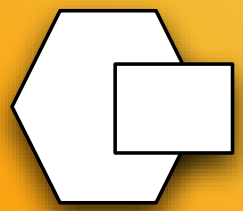


concrete pattern instances

coding c…

b…

partial and **intentional** adherence



discrepancies

separate **definition** per implementation **variant**

Software Quality Assurance

desirable properties

1

pattern definition
close to prototypical implementation
far from particular variants

2

pattern detection
non-strict
semantic

Motivating Example

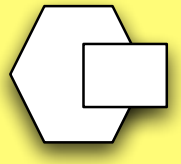
software quality property

"There should be **no direct field accesses** outside of **getter and setter methods**."

Motivating Example

software quality property
"There should be **no direct field accesses** outside of **getter and setter methods**."

```
class Y {
   private X var;

   public void setVar(X val) {
      var = val;
   }

   public X getVar {
      return var;
   }
}
```

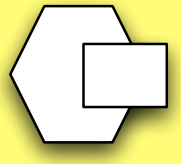


Motivating Example

software quality property

"There should be **no direct field accesses** outside of **getter and setter methods**."

```
class Y {
   private X var;

   public void setVar(X val) {
      var = val;
   }

   public X getVar {
      return var;
   }
}
```

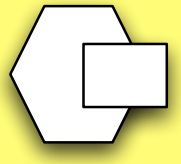


```
class Person {
  private Date birthday;
  private Integer age;
  private boolean ageDirty;

  public void setBirthday(Date newDay) {
    ageDirty = false; birthday = newDay;
  }

  public Integer getAge() {
    if(ageDirty) age = ...;
    return age;
  }
}
```

Motivating Example

software quality property
"There should be **no direct field accesses** outside of **getter and setter methods**."

```
class Y {
   private X var;

   public void setVar(X val) {
      var = val;
   }

   public X getVar {
      return var;
   }
}
```

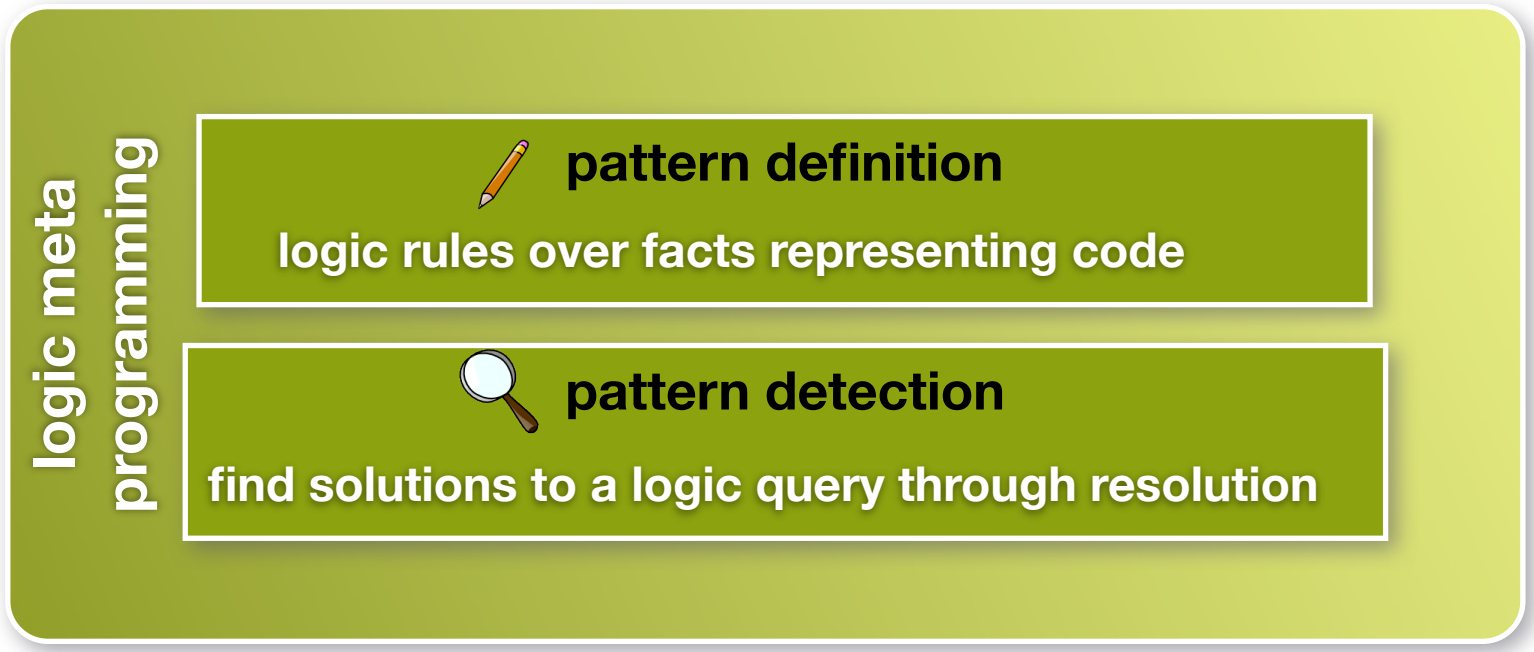


```
class Person {
  private Date birthday;
  private Integer age;
  private boolean ageDirty;

  public void setBirthday(Date newDay) {
    ageDirty = false; birthday = newDay;
  }

  public Integer getAge() {
    if(ageDirty) age = ...;
    return age;
  }
}
```

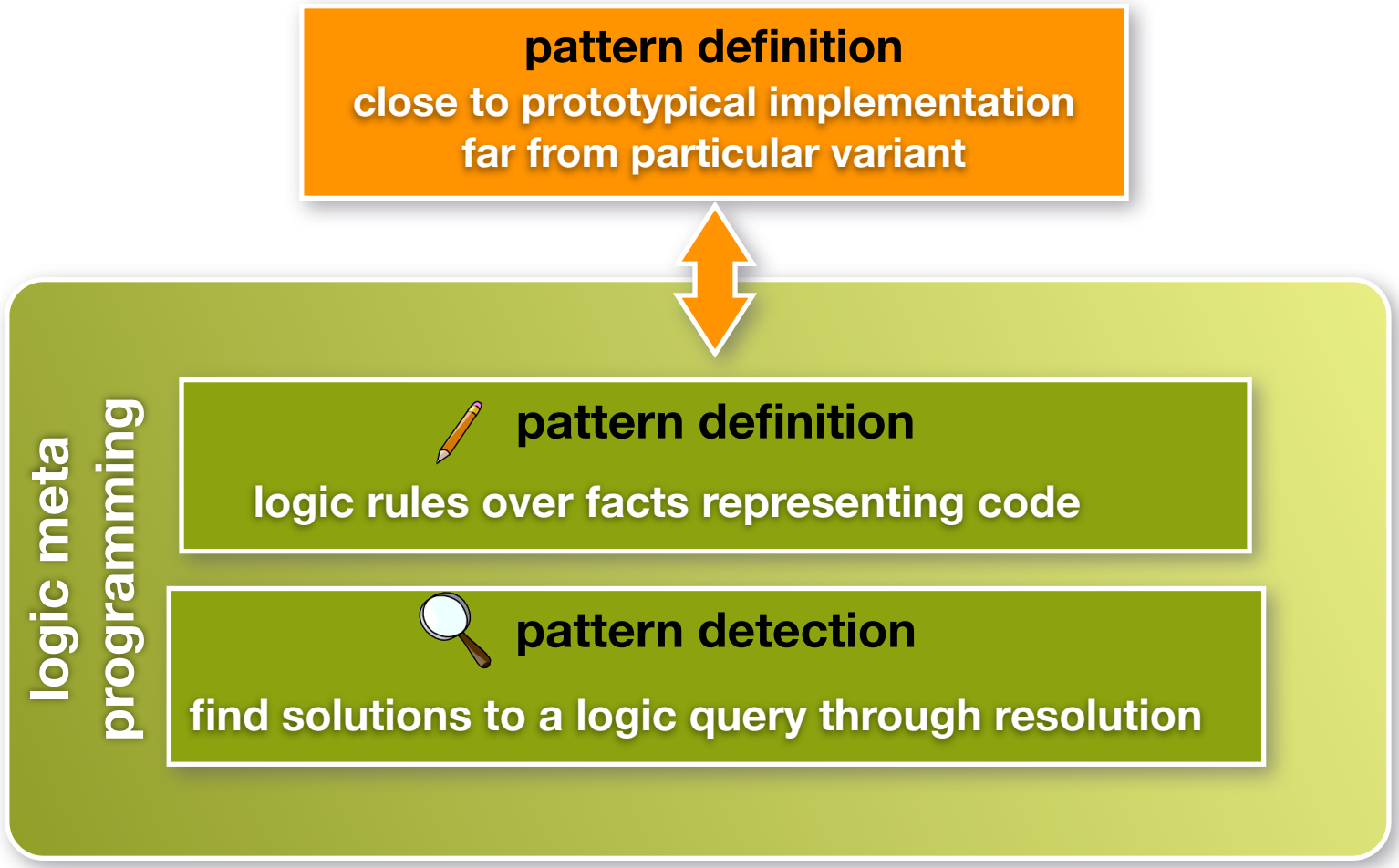
would mistakenly be reported as violating the quality property

adheres to the intention of the prototypical getter method

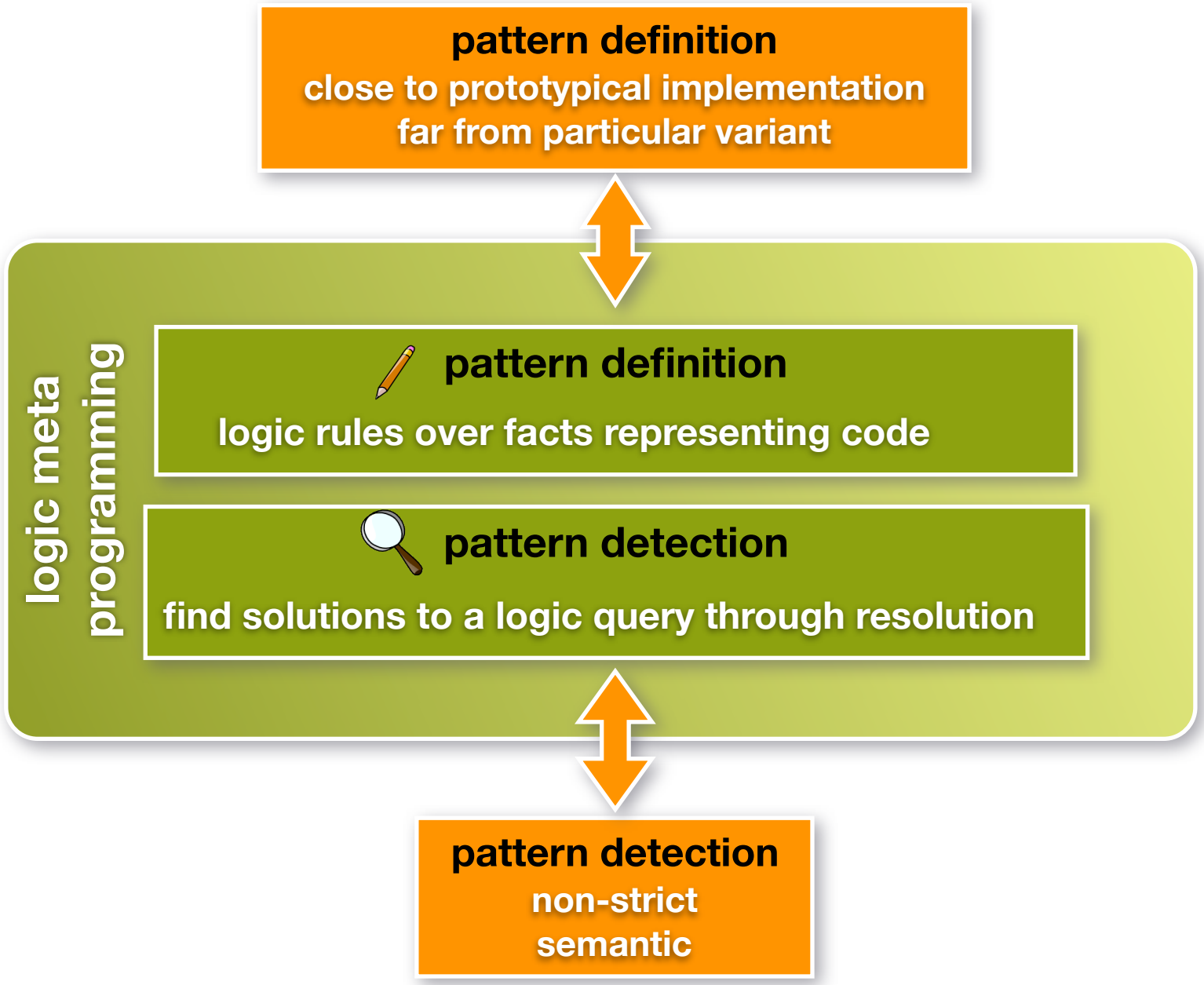
Towards a Solution



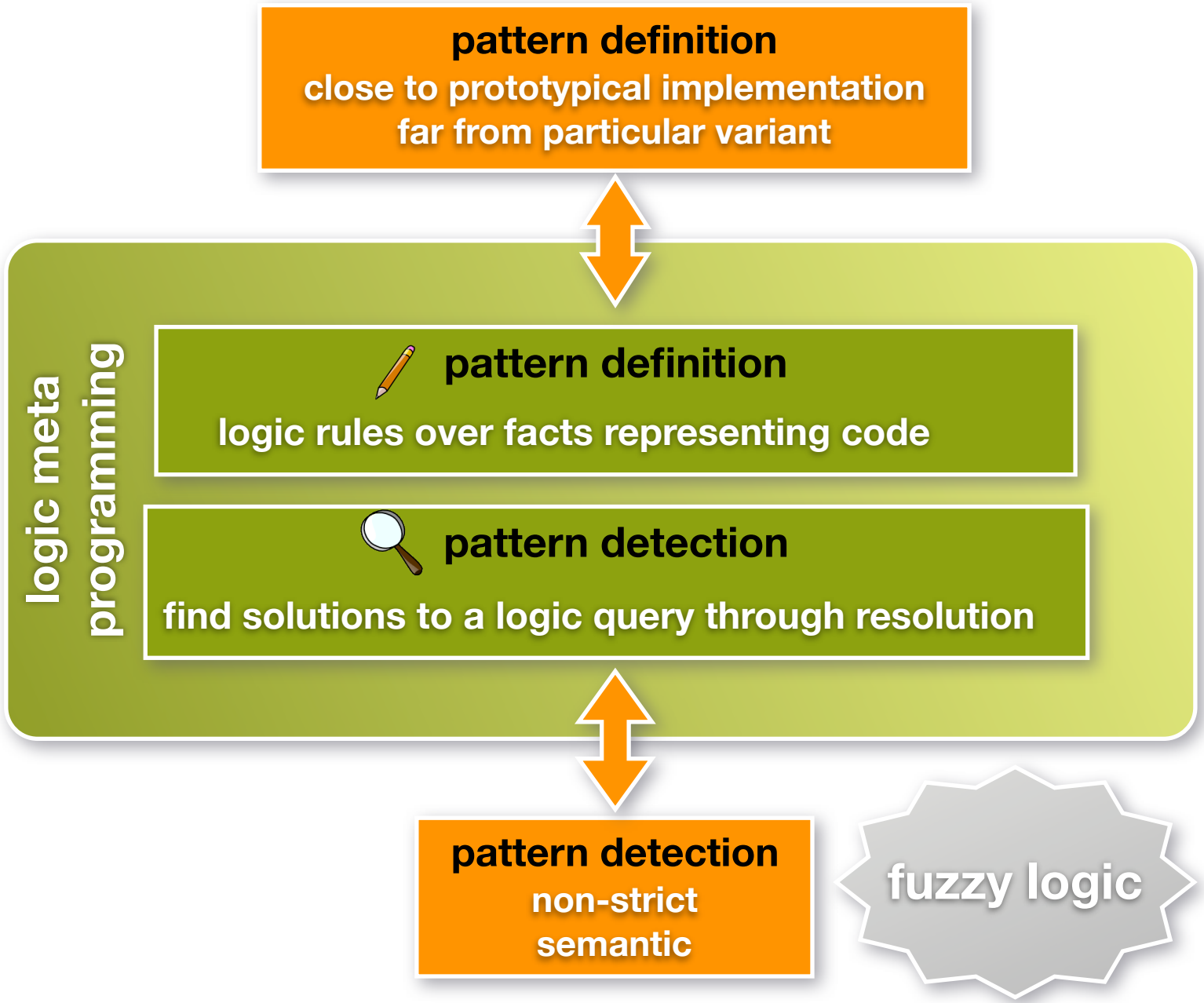
Towards a Solution



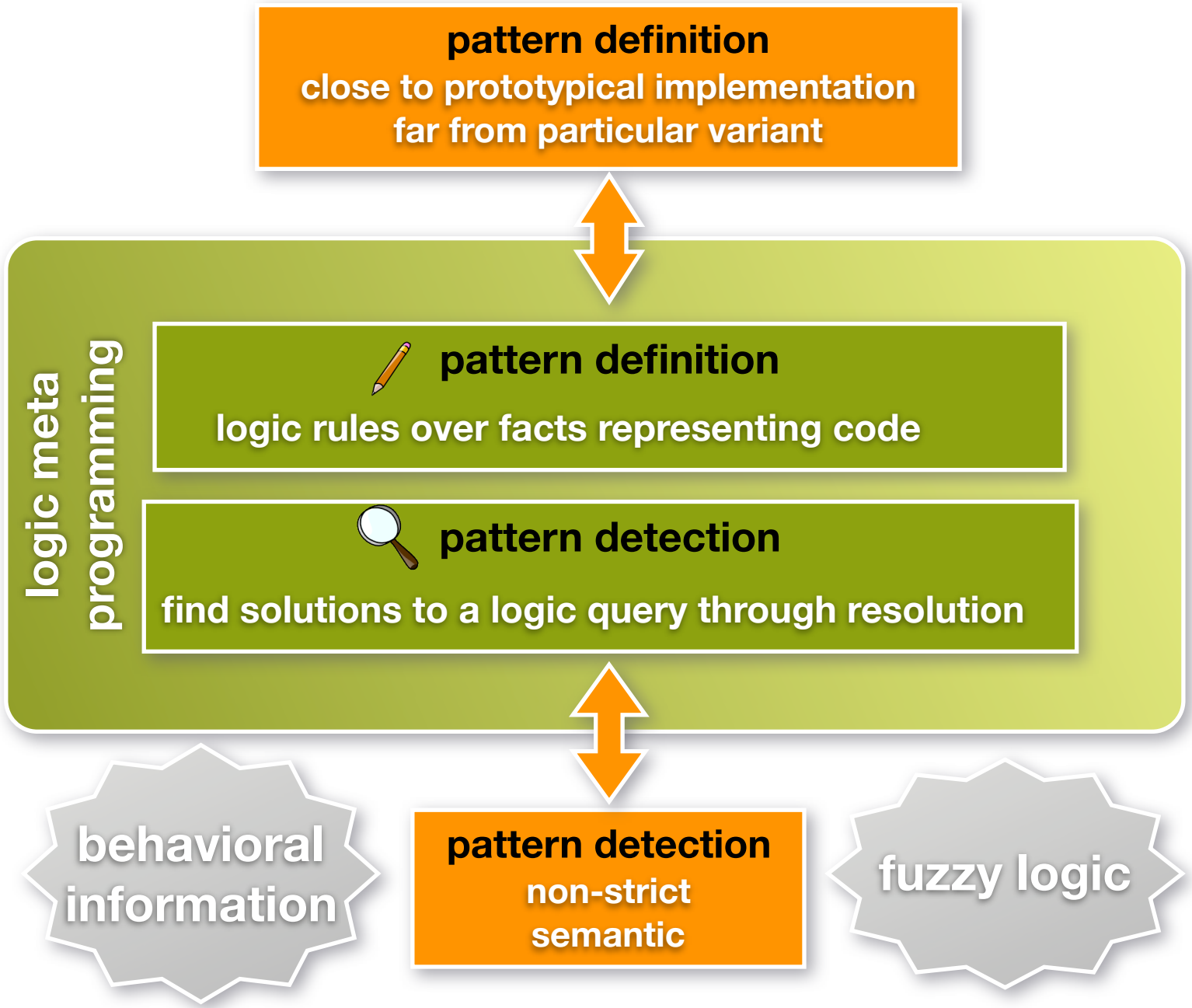
Towards a Solution



Towards a Solution



Towards a Solution



Solution Cornerstones

Solution Cornerstones

logic of quantified truth

rules annotated with partial **truth degrees**

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

fuzzy logic

Solution Cornerstones

logic of quantified truth

rules annotated with partial **truth degrees**

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

fuzzy logic

fuzzified resolution procedure

conclusions from **partially satisfied premises**

✓ *partially adhering instances*

Solution Cornerstones

logic of quantified truth

rules annotated with partial **truth degrees**

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

non-strict

fuzzy logic

fuzzified resolution procedure

conclusions from **partially satisfied premises**

✓ *partially adhering instances*

Solution Cornerstones

logic of quantified truth

rules annotated with partial **truth degrees**

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

non-strict

fuzzy logic

fuzzified resolution procedure

conclusions from **partially satisfied premises**

✓ *partially adhering instances*

behavioral information

Solution Cornerstones

logic of quantified truth

rules annotated with partial **truth degrees**

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

non-strict

fuzzy logic

fuzzified resolution procedure

conclusions from **partially satisfied premises**

✓ *partially adhering instances*

behavioral information

static analyses **approximate** actual behavior

✓ *introduces* ***partial truths***

end-users unacquainted

✓ *keep conditions over* ***source code***

static analysis

Solution Cornerstones

logic of quantified truth

rules annotated with partial **truth degrees**

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

non-strict

fuzzy logic

fuzzified resolution procedure

conclusions from **partially satisfied premises**

✓ *partially adhering instances*

unify **syntactically different** expressions

⇔ might **evaluate** to **same object**

✓ *conditions over code interpreted as conditions over the behavior it implements*

fuzzified unification procedure

behavioral information

static analyses **approximate** actual behavior

✓ *introduces* ***partial truths***

end-users unacquainted

✓ *keep conditions over* ***source code***

static analysis

Solution Cornerstones

logic of quantified truth

rules annotated with partial **truth degrees**

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

non-strict

fuzzy logic

fuzzified resolution procedure

conclusions from **partially satisfied premises**

✓ *partially adhering instances*

unify **syntactically different** expressions

⇔ might **evaluate** to **same object**

✓ *conditions over code interpreted as conditions over the behavior it implements*

semantic

behavioral information

fuzzified unification procedure

static analyses **approximate** actual behavior

✓ *introduces* ***partial truths***

end-users unacquainted

✓ *keep conditions over* ***source code***

static analysis

In Detail: Logic Meta Programming

Soul [2001:wuyts]

Irish [2004:fabry]

Java

```
class Bar {
    float x;
    public void foo() {
      x = 42 + 3.14d;
    }
}
```

In Detail: Logic Meta Programming

Soul [2001:wuyts]

Irish [2004:fabry]

Java

```
class Bar {
    float x;
    public void foo() {
      x = 42 + 3.14d;
    }
}
```

Logic representation

```
methodStatements(?m,?s),
?s = statements(<assign('float', '=',
                  variable('float', 'x'),
                  binaryExp('double','+',
                     literal('int', '42'),
                     literal('double', '3.14d')))>)
```

In Detail: Logic Meta Programming

```
assign(?type, ?operator, ?lhs, ?rhs)
for(?init, ?expression, ?update, ?body)
...
```

meta-model

Soul [2001:wuyts]

Irish [2004:fabry]

Java

```
class Bar {
    float x;
    public void foo() {
      x = 42 + 3.14d;
    }
}
```

Logic representation

```
methodStatements(?m,?s),
?s = statements(<assign('float', '=',
                  variable('float', 'x'),
                  binaryExp('double','+',
                     literal('int', '42'),
                     literal('double', '3.14d')))>)
```

In Detail: Logic Meta Programming

Soul [2001:wuyts]

Irish [2004:fabry]

```
assign(?type, ?operator, ?lhs, ?rhs)
for(?init, ?expression, ?update, ?body)
...
```

meta-model

Java

```
class Bar {
    float x;
    public void foo() {
      x = 42 + 3.14d;
    }
}
```

Logic representation

```
methodStatements(?m,?s),
?s = statements(<assign('float', '=',
                  variable('float', 'x'),
                  binaryExp('double','+',
                     literal('int', '42'),
                     literal('double', '3.14d')))>)
```

Logic rules

```
isInHierarchyOf(?directSubclass, ?root) if
  isSubClassOf(?directSubclass, ?root)

isInHierarchyOf(?indirectSubclass, ?root) if
  isSubClassOf(?indirectSubclass, ?parent),
  isInHierarchyOf(?parent, ?root)
```

In Detail: Logic Meta Programming

Soul [2001:wuyts]

Irish [2004:fabry]

```
assign(?type, ?operator, ?lhs, ?rhs)
for(?init, ?expression, ?update, ?body)
...
```

meta-model

Java

```
class Bar {
    float x;
    public void foo() {
      x = 42 + 3.14d;
    }
}
```

Logic representation

```
methodStatements(?m,?s),
?s = statements(<assign('float', '=',
                  variable('float', 'x'),
                  binaryExp('double','+',
                     literal('int', '42'),
                     literal('double', '3.14d')))>)
```

verify whether
`testapp.ComponentVisitor`
extends `java.lang.Object`

Logic query

Logic rules

```
isInHierarchyOf(?directSubclass, ?root) if
  isSubClassOf(?directSubclass, ?root)

isInHierarchyOf(?indirectSubclass, ?root) if
  isSubClassOf(?indirectSubclass, ?parent),
  isInHierarchyOf(?parent, ?root)
```

find all subclasses of
`java.lang.Object`

Logic query

In Detail: Fuzzy Logic Programming

LP with quantified truth

weighted logic rules

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

fuzzy resolution procedure

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

similar to f-Prolog [1990:liu]

In Detail: Fuzzy Logic Programming

LP with quantified truth

weighted logic rules

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

fuzzy resolution procedure

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

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

similar to f-Prolog [1990:liu]

In Detail: Fuzzy Logic Programming

LP with quantified truth

weighted logic rules

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

fuzzy resolution procedure

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

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

similar to f-Prolog [1990:liu]

```
sold(flowers, 15).
attractive_packaging(chips) : 0.9.
well_advertised(chips) : 0.6.

popular_product(?product) if
  sold(?product, ?amount),
  ?amount > 10.

popular_product(?product) : 0.8 if
  attractive_packaging(?product),
  well_advertised(?product).
```

In Detail: Fuzzy Logic Programming

LP with quantified truth

weighted logic rules

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

fuzzy resolution procedure

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

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

similar to f-Prolog [1990:liu]

if popular_product(?p) : *?c*

?p	?c
flowers	1
chips	min(0.9, 0.6)*0.8 = 0.48

```
sold(flowers, 15).
attractive_packaging(chips) : 0.9.
well_advertised(chips) : 0.6.

popular_product(?product) if
  sold(?product, ?amount),
  ?amount > 10.

popular_product(?product) : 0.8 if
  attractive_packaging(?product),
  well_advertised(?product).
```

In Detail: Similarity-Based Unification

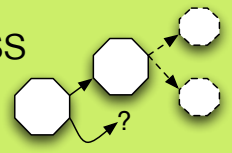
fuzzified unification procedure

unifies **incompatible** logic terms ⇔ considered **similar** to an extent

need similarity **relation** between terms

incorporate **behavioral** information into reasoning process

points-to analysis



approximates set of **objects** a reference **might** point to at run-time

In Detail: Similarity-Based Unification

fuzzified unification procedure

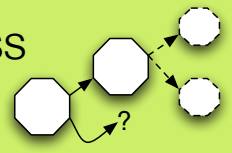
unifies **incompatible** logic terms ⇔ considered **similar** to an extent

need similarity **relation** between terms

incorporate **behavioral** information into reasoning process

points-to analysis

approximates set of **objects** a reference **might** point to at run-time



fuzzy unification of **logic terms** representing **parse tree node**

- ✓ functor representation **syntactically equivalent**
  succeed with truth degree **1**
- ✓ **might** evaluate to **overlapping sets of objects** at run-time
  succeed with truth degree **0.5**
- ✓ **fail**

In Detail: Similarity-Based Unification

fuzzified unification procedure

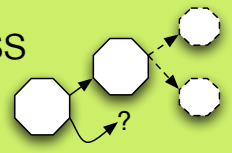
unifies **incompatible** logic terms ⇔ considered **similar** to an extent

need similarity **relation** between terms

incorporate **behavioral** information into reasoning process

points-to analysis

approximates set of **objects** a reference **might** point to at run-time



fuzzy unification of **logic terms** representing **parse tree node**

- ✓ functor representation **syntactically equivalent**
  succeed with truth degree **1**
- ✓ **might** evaluate to **overlapping sets of objects** at run-time
  succeed with truth degree **0.5**
- ✓ **fail**

```
instanceVariableInClass(?instvar, ?class),
returnStatement(?statement, ?instvar)
```

In Detail: Similarity-Based Unification

fuzzified unification procedure

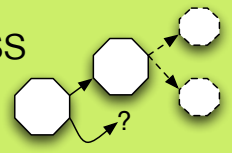
unifies **incompatible** logic terms ⇔ considered **similar** to an extent

need similarity **relation** between terms

incorporate **behavioral** information into reasoning process

points-to analysis

approximates set of **objects** a reference **might** point to at run-time



fuzzy unification of **logic terms** representing **parse tree node**

- ✓ functor representation **syntactically equivalent**
  succeed with truth degree **1**
- ✓ **might** evaluate to **overlapping sets of objects** at run-time
  succeed with truth degree **0.5**
- ✓ **fail**

```
instanceVariableInClass(?instvar, ?class),
returnStatement(?statement, ?instvar)
```

constraint over run-time values returned by statement

Getter Method Best Practice Pattern

```
class Y {
 private X var;

 public X getVar {
      return var;
   }
}
```

Getter Method Best Practice Pattern

```
class Y {
 private X var;

 public X getVar {
      return var;
   }
}
```

```
getterMethod(?class, ?method, ?instvar) if
  methodInClass(?method, ?class),
  instanceVariableInClassChain(?instvar, ?class),
  variableName(?instvar, ?vname),
  methodStatements(?method,
                   <return(variable(?vtype, ?vname))>)
```

Getter Method Best Practice Pattern

```
class Y {
 private X var;

 public X getVar {
      return var;
   }
}
```

```
getterMethod(?class, ?method, ?instvar) if
  methodInClass(?method, ?class),
  instanceVariableInClassChain(?instvar, ?class),
  variableName(?instvar, ?vname),
  methodStatements(?method,
                   <return(variable(?vtype, ?vname))>)
```

= 1

```
public Integer getSum() {
   return sum;
}
```

Getter Method Best Practice Pattern

```
class Y {
 private X var;

 public X getVar {
      return var;
   }
}
```

```
getterMethod(?class, ?method, ?instvar) if
  methodInClass(?method, ?class),
  instanceVariableInClassChain(?instvar, ?class),
  variableName(?instvar, ?vname),
  methodStatements(?method,
                   <return(variable(?vtype, ?vname))>)
```

```
getterMethod(?class, ?method, ?instvar) : 0.9 if
  methodInClass(?method, ?class),
  instanceVariableInClassChain(?instvar, ?class),
  statementInMethod(?statement, ?method),
  returnStatement(?statement, ?instvar).
```

= 1

```
public Integer getSum() {
   return sum;
}
```

Getter Method Best Practice Pattern

```
class Y {
 private X var;

 public X getVar {
      return var;
   }
}
```

```
getterMethod(?class, ?method, ?instvar) if
  methodInClass(?method, ?class),
  instanceVariableInClassChain(?instvar, ?class),
  variableName(?instvar, ?vname),
  methodStatements(?method,
                   <return(variable(?vtype, ?vname))>)
```

= 1

```
getterMethod(?class, ?method, ?instvar) : 0.9 if
  methodInClass(?method, ?class),
  instanceVariableInClassChain(?instvar, ?class),
  statementInMethod(?statement, ?method),
  returnStatement(?statement, ?instvar).
```

0.9 =

```
public Integer getSum() {
   return sum;
}
```

Getter Method Best Practice Pattern

```
class Y {
 private X var;

 public X getVar {
      return var;
   }
}
```

```
getterMethod(?class, ?method, ?instvar) if
  methodInClass(?method, ?class),
  instanceVariableInClassChain(?instvar, ?class),
  variableName(?instvar, ?vname),
  methodStatements(?method,
                   <return(variable(?vtype, ?vname))>)
```

```
getterMethod(?class, ?method, ?instvar) : 0.9 if
  methodInClass(?method, ?class),
  instanceVariableInClassChain(?instvar, ?class),
  statementInMethod(?statement, ?method),
  returnStatement(?statement, ?instvar).
```

```
public Integer returnSum() {
   Integer val = (Integer) indirectReturnOfArgument(sum, 10);
   return val;
}

public Object indirectReturnOfArgument(Object o, int delay) {
   if(delay == 0)
      return o;
   else
      return indirectReturnOfArgument(o, delay - 1);
}
```

implements intention of pattern

Getter Method Best Practice Pattern

```
class Y {
 private X var;

 public X getVar {
      return var;
   }
}
```

```
getterMethod(?class, ?method, ?instvar) if
  methodInClass(?method, ?class),
  instanceVariableInClassChain(?instvar, ?class),
  variableName(?instvar, ?vname),
  methodStatements(?method,
                   <return(variable(?vtype, ?vname))>)
```

```
getterMethod(?class, ?method, ?instvar) : 0.9 if
  methodInClass(?method, ?class),
  instanceVariableInClassChain(?instvar, ?class),
  statementInMethod(?statement, ?method),
  returnStatement(?statement, ?instvar).
```

0.45

```
public Integer returnSum() {
   Integer val = (Integer) indirectReturnOfArgument(sum, 10);
   return val;
}

public Object indirectReturnOfArgument(Object o, int delay) {
   if(delay == 0)
      return o;
   else
      return indirectReturnOfArgument(o, delay - 1);
}
```

```
val ~(?instvar=sum)
```

implements intention of pattern

Getter Method Best Practice Pattern

```
class Y {
 private X var;

 public X getVar {
      return var;
   }
}
```

```
getterMethod(?class, ?method, ?instvar) if
  methodInClass(?method, ?class),
  instanceVariableInClassChain(?instvar, ?class),
  variableName(?instvar, ?vname),
  methodStatements(?method,
                   <return(variable(?vtype, ?vname))>)
```

```
getterMethod(?class, ?method, ?instvar) : 0.9 if
  methodInClass(?method, ?class),
  instanceVariableInClassChain(?instvar, ?class),
  statementInMethod(?statement, ?method),
  returnStatement(?statement, ?instvar).
```

Getter Method Best Practice Pattern

```
class Y {
 private X var;

 public X getVar {
      return var;
   }
}
```

```
getterMethod(?class, ?method, ?instvar) if
  methodInClass(?method, ?class),
  instanceVariableInClassChain(?instvar, ?class),
  variableName(?instvar, ?vname),
  methodStatements(?method,
                   <return(variable(?vtype, ?vname))>)
```

```
getterMethod(?class, ?method, ?instvar) : 0.9 if
  methodInClass(?method, ?class),
  instanceVariableInClassChain(?instvar, ?class),
  statementInMethod(?statement, ?method),
  returnStatement(?statement, ?instvar).
```

```
public Integer retrieveSum() {
   Object retrieved = returnSum();
   if(retrieved instanceof Integer) {
      Integer value = (Integer) retrieved;
      return value;
   } else return getSum();
}
```

implements intention of pattern

Getter Method Best Practice Pattern

```
class Y {
 private X var;

 public X getVar {
      return var;
   }
}
```

```
getterMethod(?class, ?method, ?instvar) if
  methodInClass(?method, ?class),
  instanceVariableInClassChain(?instvar, ?class),
  variableName(?instvar, ?vname),
  methodStatements(?method,
                   <return(variable(?vtype, ?vname))>)
```

```
getterMethod(?class, ?method, ?instvar) : 0.9 if
  methodInClass(?method, ?class),
  instanceVariableInClassChain(?instvar, ?class),
  statementInMethod(?statement, ?method),
  returnStatement(?statement, ?instvar).
```

value ~(?instvar=sum) 0.45

0.45 getSum() ~(?instvar=sum)

```
public Integer retrieveSum() {
   Object retrieved = returnSum();
   if(retrieved instanceof Integer) {
      Integer value = (Integer) retrieved;
      return value;
   } else return getSum();
}
```

implements intention of pattern

Correctly Named Getter Method

```
correctlyNamedGetterMethod(?c, ?m, ?var) if
  getterMethod(?c, ?m, ?var) : ?c1,
  selectorOfMethod(?selector, ?m),
  instanceVariableName(?var, ?varname),
  capitalized(?varname, ?cvarname),
  concat('get', ?cvarname, ?sel),
  similar(?sel, ?selector) : ?c2,
  ?c1 * ?c2.
```

?m is a getter for *?var* with truth degree *?c1*

its selector is **similar** to 'getVar' with truth degree *?c2*

truth degree of this condition is *?c1 * ?c2*

Correctly Named Getter Method

```
correctlyNamedGetterMethod(?c, ?m, ?var) if
  getterMethod(?c, ?m, ?var) : ?c1,
  selectorOfMethod(?selector, ?m),
  instanceVariableName(?var, ?varname),
  capitalized(?varname, ?cvarname),
  concat('get', ?cvarname, ?sel),
  similar(?sel, ?selector) : ?c2,
  ?c1 * ?c2.
```

?m is a getter for *?var* with truth degree *?c1*

its selector is **similar** to 'getVar' with truth degree *?c2*

truth degree of this condition is *?c1 * ?c2*

similar(*?s1*, *?s2*)

$$1 - \frac{e(?s1, ?s2)}{max(|?s1|, |?s2|)}$$

where e = Levensthein distance

Correctly Named Getter Method

```
correctlyNamedGetterMethod(?c, ?m, ?var) if
  getterMethod(?c, ?m, ?var) : ?c1,
  selectorOfMethod(?selector, ?m),
  instanceVariableName(?var, ?varname),
  capitalized(?varname, ?cvarname),
  concat('get', ?cvarname, ?sel),
  similar(?sel, ?selector) : ?c2,
  ?c1 * ?c2.
```

?m is a getter for *?var* with truth degree *?c1*

its selector is **similar** to 'getVar' with truth degree *?c2*

truth degree of this condition is *?c1 * ?c2*

similar(*?s1*, *?s2*)

$$1 - \frac{e(?s1, ?s2)}{max(|?s1|, |?s2|)}$$

where e = Levensthein distance

```
if correctlyNamedGetterMethod(?c, ?m, ?var) : ?t
```

?c	*?var*	*?m*	*?t*
SumCmpntVistor	sum	getSum()	1
SumCmpntVistor	sum	getSum()	0.9
SumCmpntVistor	sum	getSum()	0.45
SumCmpntVistor	sum	returnSum()	0.25
SumCmpntVistor	sum	retrieveSum()	0.20
SumCmpntVistor	sum	retrieveSum()	0.20

Correctly Named Getter Method

```
correctlyNamedGetterMethod(?c, ?m, ?var) if
  getterMethod(?c, ?m, ?var) : ?c1,
  selectorOfMethod(?selector, ?m),
  instanceVariableName(?var, ?varname),
  capitalized(?varname, ?cvarname),
  concat('get', ?cvarname, ?sel),
  similar(?sel, ?selector) : ?c2,
  ?c1 * ?c2.
```

?m is a getter for *?var* with truth degree *?c1*

its selector is **similar** to 'getVar' with truth degree *?c2*

truth degree of this condition is *?c1 * ?c2*

similar(*?s1*, *?s2*)

$$1 - \frac{e(?s1, ?s2)}{max(|?s1|, |?s2|)}$$

where e = Levensthein distance

`if correctlyNamedGetterMethod(?c, ?m, ?var) : ?t`

?c	*?var*	*?m*	*?t*
SumCmpntVistor	sum	getSum()	1
SumCmpntVistor	sum	getSum()	0.9
SumCmpntVistor	sum	getSum()	0.45
SumCmpntVistor	sum	returnSum()	0.25
SumCmpntVistor	sum	retrieveSum()	0.20
SumCmpntVistor	sum	retrieveSum()	0.20

vague classification boundaries

Setter Method Best Practice Pattern

```
class Y {
  private X var;
  public void setVar(X val) {
    var = val;
  }
}
```

Setter Method Best Practice Pattern

```
class Y {
  private X var;
  public void setVar(X val) {
    var = val;
  }
}
```

```
setterMethod(?class, ?method, ?instvar) if
  methodInClass(?method, ?class),
  argumentOfMethod(?argument, ?method),
  instanceVariableInClassChain(?instvar, ?class),
  variableName(?instvar, ?ivarname),
  methodStatements(?method, ?s),
  ?s = <assign(?atype, ?aoperator,
               variable(?lhstype, ?ivarname),
               variable(?rhstype, ?argname))>.
```

Setter Method Best Practice Pattern

```
class Y {
  private X var;
  public void setVar(X val) {
    var = val;
  }
}
```

```
setterMethod(?class, ?method, ?instvar) if
  methodInClass(?method, ?class),
  argumentOfMethod(?argument, ?method),
  instanceVariableInClassChain(?instvar, ?class),
  variableName(?instvar, ?ivarname),
  methodStatements(?method, ?s),
  ?s = <assign(?atype, ?aoperator,
               variable(?lhstype, ?ivarname),
               variable(?rhstype, ?argname))>.
```

```
setterMethod(?class, ?method, ?instvar, ?c) : 0.9 if
  methodInClass(?method, ?class),
  instanceVariableInClassChain(?instvar, ?class),
  argumentOfMethod(?argument, ?method),
  expressionInMethod(?expression, ?method),
  isAssignment(?expression, ?instvar, ?argument).
```

Setter Method Best Practice Pattern

```
class Y {
  private X var;
  public void setVar(X val) {
    var = val;
  }
}
```

```
setterMethod(?class, ?method, ?instvar) if
  methodInClass(?method, ?class),
  argumentOfMethod(?argument, ?method),
  instanceVariableInClassChain(?instvar, ?class),
  variableName(?instvar, ?ivarname),
  methodStatements(?method, ?s),
  ?s = <assign(?atype, ?aoperator,
               variable(?lhstype, ?ivarname),
               variable(?rhstype, ?argname))>.
```

```
setterMethod(?class, ?method, ?instvar, ?c) : 0.9 if
  methodInClass(?method, ?class),
  instanceVariableInClassChain(?instvar, ?class),
  argumentOfMethod(?argument, ?method),
  expressionInMethod(?expression, ?method),
  isAssignment(?expression, ?instvar, ?argument).
```

```
public void updateSum(Integer newValue)  {
    Integer int1 = new Integer(1);
    Integer int2 = new Integer(2);

    Integer[] arrayOfInts = { int1, int2, int1, int2, int1, int2};
    arrayOfInts[5] = newValue;
    for (int i = 0; i < arrayOfInts.length; i++) {
       sum = arrayOfInts[i];
    }
}
```

implements intention of pattern

Setter Method Best Practice Pattern

```
class Y {
  private X var;
  public void setVar(X val) {
    var = val;
  }
}
```

```
setterMethod(?class, ?method, ?instvar) if
  methodInClass(?method, ?class),
  argumentOfMethod(?argument, ?method),
  instanceVariableInClassChain(?instvar, ?class),
  variableName(?instvar, ?ivarname),
  methodStatements(?method, ?s),
  ?s = <assign(?atype, ?aoperator,
               variable(?lhstype, ?ivarname),
               variable(?rhstype, ?argname))>.
```

```
setterMethod(?class, ?method, ?instvar, ?c) : 0.9 if
  methodInClass(?method, ?class),
  instanceVariableInClassChain(?instvar, ?class),
  argumentOfMethod(?argument, ?method),
  expressionInMethod(?expression, ?method),
  isAssignment(?expression, ?instvar, ?argument).
```

0.45

```
arrayOfInts[i] ~(?argument=newValue)
```

implements intention of pattern

```
public void updateSum(Integer newValue)  {
    Integer int1 = new Integer(1);
    Integer int2 = new Integer(2);

    Integer[] arrayOfInts = { int1, int2, int1, int2, int1, int2};
    arrayOfInts[5] = newValue;
    for (int i = 0; i < arrayOfInts.length; i++) {
       sum = arrayOfInts[i];
    }
}
```

Conclusions

combined in an **end-user transparant** way

- logic **meta** programming
- **fuzzy** logic
- heavy-weight **static analysis** techniques

extended classical logic meta programming setup

- partial **truth degrees**
  - annotate rules with **confidence** in employed heuristics
  - express patterns with **vague** classification boundaries
- fuzzified **resolution** procedure
  - **non-strict** detection of instances partially adhering to conditions
- fuzzified **unification** procedure
  - takes **behavioral information** into account
  - **semantic** interpretation of conditions over code as conditions over the behavior it implements

quite unique

The End

Questions?

one subtlety

```
class Foo {
  public Integer getSum() {
    return sum;
  }
}

if statementInMethod(?s, ?),
   ?s = return(variable(?type, 'sum')),
   surroundingMethodName(?s, ?methodName)

Solutions: ?methodName -> 'getSum'
           ?type -> 'java.lang.Integer'
```

one subtlety

```
class Foo {
  public Integer getSum() {
    return sum;
  }
}
```

parse
tree node logic terms

can **unify** with logic functor
can be **queried** without
tree walk

```
if statementInMethod(?s, ?),
   ?s = return(variable(?type, 'sum')),
   surroundingMethodName(?s, ?methodName)

Solutions: ?methodName -> 'getSum'
           ?type -> 'java.lang.Integer'
```