Research Topics in Software Quality

Detecting Structural (Anti-)Patterns using Logic Program Queries
Applicative meta-programming

... against Eclipse JDT using Clojure

Lisp on JVM

(defn reduce-projects!
  "Clojure reduce over all projects in the workspace."
  [f initval projects]
  (reduce
    (fn [sofar p]
      (workspace-project-open! p)
      (workspace-wait-for-builds-to-finish)
      (let [result (f sofar p)]
        (workspace-project-close! p)
        result))
    initval
    projects))

=> (reduce-projects! (fn [sofar p] (conj sofar (.getName p)))
    []
    (workspace-projects))
["AmbientTalk20080201" "DesignPatterns"
 "JHotDraw51" "JavaTemplatesUnitTests" "MLIForSootUnitTests" "jEdit"]

but what if we want to know more than the name of each project?
Common meta-programming problem
... identify code with characteristics of interest

e.g. structural

class Suspicious {
    int hashCode() {
        return ...;
    }
}

control flow, data flow

scanner = new Scanner();
...
scanner.close();
...
x.next();
Applicative meta-programming
... against Eclipse JDT using Clojure
- rich in information: ASTs, structure, semantics, ...

but implementing a code search as an AST descent is tedious ....
Applicative meta-programming
... against Eclipse JDT using Clojure

package org.eclipse.jdt.core.dom

class ReturnStatement
extends Statement

Return statement AST node type.

ReturnStatement:
    return [ Expression ];

Since:
    2.0

Restriction:
    This class is not intended to be instantiated by clients.

Field Summary

<table>
<thead>
<tr>
<th>static ChildPropertyDescriptor</th>
<th>PROPERTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression_property</td>
<td>The &quot;expression&quot; structural property of this node type (child type: Expression).</td>
</tr>
</tbody>
</table>
Applicative logic meta-programming

constraints on values for ?inv and ?exp

pairs of values satisfying constraints
Building Development Tools Interactively using the EKEKO Meta-Programming Library

Coen De Roover
Vrije Universiteit Brussel, Belgium
Email: cderoove@vub.ac.be

Building Development Tools Interactively

ECLIPSE

EKEKO

Eclipse plugin

causally connected

Ekeko

meta-programming library for Clojure

logic meta-programming

specify code characteristics declaratively, leave search to logic engine

applicative meta-programming

script queries over workspace

manipulate workspace

applications

program and corpus analysis

program transformation

tool building

7

[16]...
Think of it as querying a database of program information!
Logic relations: ast/2 and has/3 “tables”

### (ast ?type ?node)

<table>
<thead>
<tr>
<th>?type</th>
<th>?node</th>
</tr>
</thead>
<tbody>
<tr>
<td>SingleVariableDeclaration</td>
<td>EntityIdentifier qualifier ...</td>
</tr>
<tr>
<td>SimpleType</td>
<td>EntityIdentifier ...</td>
</tr>
<tr>
<td>SimpleName</td>
<td>java ...</td>
</tr>
<tr>
<td>ReturnStatement</td>
<td>return body; ...</td>
</tr>
<tr>
<td>ReturnStatement</td>
<td>return type; ...</td>
</tr>
<tr>
<td>SimpleName</td>
<td>lang ...</td>
</tr>
<tr>
<td>:ReturnStatement</td>
<td>return value instanceof EntityIdentifier; ...</td>
</tr>
<tr>
<td>:ReturnStatement</td>
<td>return email_addresses; ...</td>
</tr>
<tr>
<td>:SimpleType</td>
<td>Type ...</td>
</tr>
</tbody>
</table>

#### relation between a ?node of an Abstract Syntax Tree (AST) and its ?type

### (has ?property ?node ?value)

<table>
<thead>
<tr>
<th>?property</th>
<th>?node</th>
<th>?value</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>assert (descriptor.canBeAssigned(value));</td>
<td>null</td>
</tr>
<tr>
<td>expression</td>
<td>assert (descriptor.canBeAssigned(value));</td>
<td>(descriptor.canBeAssigned(value))</td>
</tr>
<tr>
<td>:rightHandSide</td>
<td>this.typeArguments=typeArguments</td>
<td>typeArguments</td>
</tr>
<tr>
<td>:operator</td>
<td>this.typeArguments=typeArguments</td>
<td></td>
</tr>
<tr>
<td>:leftHandSide</td>
<td>this.name=name</td>
<td></td>
</tr>
<tr>
<td>:rightHandSide</td>
<td>this.name=name</td>
<td></td>
</tr>
<tr>
<td>:leftHandSide</td>
<td>this.owningClass=owningClass</td>
<td></td>
</tr>
<tr>
<td>:rightHandSide</td>
<td>this.owningClass=owningClass</td>
<td></td>
</tr>
<tr>
<td>statements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:rightHandSide</td>
<td>this.locationInParent=locationInParent</td>
<td></td>
</tr>
</tbody>
</table>

#### relation of AST nodes and the values of their properties
Logic querying: AST relations

SELECT ast.node as ?statement, has.value as ?expression
FROM ast, has
WHERE ast.type = :ReturnStatement AND has.property = :expression AND has.node = ast.node;

relation of all AST nodes of type :ReturnStatement and the value of their :expression property

equivalent logic query

(ekeko [?statement ?expression]
  (ast :ReturnStatement ?statement)
  (has :expression ?statement ?expression))
Logic programming: defining relations

defining relation:  
    (defn  
      expression|returned  
        [?expression]  
          (fresh [?statement]  
              (ast :ReturnStatement ?statement)  
                (has :expression ?statement ?expression))))  

using newly defined relation:  
    (ekeko* [?returned ?type]  
        (expression|returned ?returned)  
          (ast|expression-type ?returned ?type)  
            (type|binary ?type))

returned expressions of a type defined in compiled rather than source code
Applicative logic meta-programming

... core.logic in a nutshell

embedding of logic programming

port of Kanren [Friedman, Kiselyov, Bird] to Clojure by David Nolen

no operational impurities (e.g., cut), yet high performance

features tabling, extensible constraints, extensible unification protocols

logic goals

functions that take a substitution

either return a possibly infinite stream of substitutions (success) or nil (failure)

constructors:

always success: s#, always failure: f#, success if arguments unify:  ==

composing goals

introduces lexically scoped variables

chains goals together

interleaves substitutions from goals

abstraction

(fresh [?x ?y]
  (== ?x ?y)
  (== ?x 5)))

(fresh [?x ?y]
  (conde
   [(== ?x 1) ..]
   [(== ?x 5) ..]))

(defn g [?y]
  (fresh []
    (== ?y 5)))

(fresh [?x]
  (g ?x))
Ekeko Library: relations for logic meta-programming

**Syntactic**

```java
for (Object i : collection)
```

**Structural**

```java
class Ouch {
    int hashCode() {
        return ...;
    }
}
```

**Control and Data Flow**

```java
scanner = new Scanner();
...
x.close();
...
scanner.next();
```

```
(\text{ast} \ ?\text{kind} \ ?\text{ast})
(\text{has} \ ?\text{property} \ ?\text{ast} \ ?\text{value})
(\text{ast-encompassing}|\text{method}+ \ ?\text{ast} \ ?\text{m})
(\text{ast-encompassing}|\text{type}+ \ ?\text{ast} \ ?\text{t})
```

```
(\text{classfile-type} \ ?\text{binaryfile} \ ?\text{type})
(\text{type-type}|\text{sub}+ \ ?\text{type} \ ?\text{subtype})
(\text{type-name}|\text{qualified} \ ?\text{type} \ ?\text{qname})
(\text{advice-shadow} \ ?\text{advice} \ ?\text{shadow})
```

```
(\text{method}|\text{soot-cfg} \ ?\text{m} \ ?\text{cfg})
(\text{unit}|\text{soot-usebox} \ ?\text{u} \ ?\text{ub})
(\text{local}|\text{soot-pointstoset} \ ?\text{l} \ ?\text{p})
(\text{soot}|\text{may} \ ?\text{l1} \ ?\text{l2})
```
Ekeko Library: functions for applicative meta-programming

rewriting

(remove-node node)
(replace-node node newnode)
(change-property node property value)
(apply-and-reset-rewrites!)

visualizing

(visualize nodes edges
 :layout layout
 :node-label label fn
 :edge-label label fn
 . . .)

tooling

(add-problem-marker marker node)
(register-quickfix marker rewrite fn)
(reduce-workspace fn initval)
(wait-for-builds-to-finish)
Ekeko Library: documentation

Namespaces

ccw.debug.server.repl
damp.ekeko
damp.ekeko.ekekomodel
damp.ekeko.gui
damp.ekeko.jdt.ast
damp.ekeko.jdt.astbindings
damp.ekeko.jdt.astnode
damp.ekeko.jdt.aststructure
damp.ekeko.jdt.bindings
damp.ekeko.jdt.convenience
damp.ekeko.jdt.javaprojectmodel
damp.ekeko.jdt.markers
damp.ekeko.jdt.rewrites
damp.ekeko.jdt.soot
damp.ekeko.jdt.structure
damp.ekeko.logic
damp.ekeko.soot.projectmodel
damp.ekeko.soot.soot
damp.ekeko.util.jobs
damp.ekeko.util.text
damp.ekeko.workspace.projectmodel
damp.ekeko.workspace.reification
damp.ekeko.workspace.workspace
damp.util.interop

Public Vars

ast
ast-location
ast-methoddeclaration:encompassing
ast-methoddeclaration:encompassing
ast-parent
ast-root
ast-typedeclaration:encompassing
ast-declaration
ast-declaration-modifier
astdealarationresolved
astexpressionreference
astfieldaccess
astinvocation
astparametername
astresolved
asttype
child
child+
has
method-cfg
method-cfg:entry
method-cfg:exit
method-statements
name-namessamelqualified

Public Vars

damp.ekeko.jdt.ast documentation

Low-level relations of JDT AST nodes.

ast

( ast : ?keyword : ?node )
Reifies the relation between ASTNode instances ?node and their kind ?keyword. In general, ?keyword is the keyword that corresponds to the capitalized, unqualified name of ?node's class.

Examples:
:: all method declarations
( eko*: ?node ) ( ast : MethodDeclaration ?node )
:: all ast nodes of any kind

See also:
API documentation of org.eclipse.jdt.core.dom.ASTNode

Source

ast-location

( ast-location : ?ast : ?locationVector )

Source

ast-methoddeclaration:encompassing

( ast-methoddeclaration:encompassing : ?ast : ?m )
Relation between ASTNode ?ast and the MethodDeclaration ?m that encompasses it. Operationally, performs a recursive ascend.

See also:
Predicate ast-encompassing-method-non-failing/2 unifies ?m with nil if there is no encompassing method.

Source
**Applicative logic meta-programming**

... in practice: building a continuous defect marker

A method returning null instead of an empty collection:

```java
public List<Integer> getChildren() {
    return null;
}
```

**vs**

```java
public List<Integer> getChildren() {
    return Collections.emptyList();
}
```

**Mark defects**

**Correct defects**

**Tools**
Applicative logic meta-programming

... in practice: building a continuous defect marker

(\texttt{ekeko*} \ [?s \ ?e]
  \texttt{(ast :ReturnStatement \ ?s)}
  \texttt{(has :expression \ ?s \ ?e)}
  \texttt{(ast :NullLiteral \ ?e)}))

\texttt{(defn statement\|returningnull} \ [?s]
  \texttt{(fresh} \ [?e]
    \texttt{(ast :ReturnStatement \ ?s)}
    \texttt{(has :expression \ ?s \ ?e)}
    \texttt{(ast :NullLiteral \ ?e)}))

\texttt{(defn method\|returningnull-returntype} \ [?method \ ?type]
  \texttt{(fresh} \ [?s \ ?t]
    \texttt{(statement\|returningnull \ ?s)}
    \texttt{(ast-methoddeclaration\|encompassing \ ?s \ ?method)}
    \texttt{(has :returnType2 \ ?method \ ?t)}
    \texttt{(ast\|type\-type \ ?t \ ?type)}))

\texttt{same, but with the return type of the enclosing method}

\texttt{supposed to subtype java.util.Collection}
Applicative logic meta-programming

... in practice: building a continuous defect marker

same, but only if method returns a Collection type

```clojure
(defn method|returningnull-returntype|collection
  [?method ?type]
  (fresh [?collectiontype]
    (method|returningnull-returntype ?method ?type)
    (conde [[(equals ?type ?collectiontype)]]
      [[(type-type|super+ ?type ?collectiontype)]]
      (type-name|qualified|string ?collectiontype "java.util.Collection")))
```
Applicative logic meta-programming

... in practice: building a continuous defect marker

```latex
(ekelo* [?m ?t] (method|returningnull-returntype|collection ?m ?t))

(def
  listener
  (model-update-listener
    (incremental-node-marker
      (fn []
        (map first
          (damp.ekelo/ekelo [?m ?t]
            (method|returningnull-returntype|collection ?m ?t))))))

(register-listener listener)
;change file
(unregister-listener listener)
```
Applicative logic meta-programming

... in practice: detecting design patterns

Composite pattern

From Wikipedia, the free encyclopedia

In software engineering, the composite pattern is a partitioning design pattern. The composite pattern describes that a group of objects is to be treated in the same way as a single instance of an object. The intent of a composite is to "compose" objects into tree structures to represent part-whole hierarchies. Implementing the composite pattern lets clients treat individual objects and compositions uniformly.[1]
Applicative logic meta-programming

... in practice: detecting design patterns

(defn name-prefix
  [?name ?prefix]
  (fresh [?string]
    (namesimple-string ?name ?string)
    (succeeds (.startsWith ?string ?prefix)))))

defn name|startingWithAdd
  [?name]
  (all
    (name-prefix ?name "add")))

defn method|startingWithAdd
  [?method]
  (fresh [?name]
    (ast :MethodDeclaration ?method)
    (has :name ?method ?name)
    (name|startingWithAdd ?name)))
Applicative logic meta-programming

... in practice: detecting design patterns

(defn invocation|onCollection
  [?invocation]
  (fresh [?receiver ?receivertype ?collectiontype]
    (ast :MethodInvocation ?invocation)
    (has :expression ?invocation ?receiver)
    (ast|expression-type ?receiver ?receivertype)
    (type-type|super+ ?receivertype ?collectiontype)
    (type-name|qualified|string ?collectiontype "java.util.Collection")))

(defn method|invokingAddOnCollection
  [?method]
  (fresh [?invocation ?name]
    (ast :MethodDeclaration ?method)
    (child+ ?method ?invocation)
    (invocation|onCollection ?invocation)
    (has :name ?invocation ?name)
    (name|simple-string ?name "add")))

transitive child relation
Applicative logic meta-programming

... in practice: detecting design patterns

```
(defn invocation|onCollection
  [?invocation]
  (fresh [?receiver ?receivertype ?collectiontype]
    (ast :MethodInvocation ?invocation)
    (has :expression ?invocation ?receiver)
    (ast|expression-type ?receiver ?receivertype)
    (type-type|super+ ?receivertype ?collectiontype)
    (type-name|qualified|string ?collectiontype "java.util.Collection")))
```

```
(defn method|invokingAddOnCollection
  [?method]
  (fresh [?invocation ?name]
    (ast :MethodDeclaration ?method)
    (child+ ?method ?invocation)
    (invocation|onCollection ?invocation)
    (has :name ?invocation ?name)
    (name|simple-string ?name "add")))
```
... in practice: detecting design patterns

(defn method-parametertypetype
  [...method ?resolvedtype]
  (fresh [...parameter ?parametertypetype]
    (ast :MethodDeclaration ?method)
    (child :parameters ?method ?parameter)
    (has :type ?parameter ?parametertypetype)
    (ast|type-type ?parametertypetype ?resolvedtype))))

(defn composite-component
  [...composite ?component]
  (fresh [...method ?invocation ?compositetype ?componenttype]
    (ast :TypeDeclaration ?composite)
    (typedeclaration-type ?composite ?compositetype)
    (child :bodyDeclarations ?composite ?method)
    (method|startingWithAdd ?method)
    (method|invokingAddOnCollection ?method)
    (method-parametertypetype ?method ?componenttype)
    (typedeclaration-type ?component ?componenttype)
    (type-type|super+ ?compositetype ?componenttype))))
More advanced declarative tools (1)

... QwalKeko: querying a program’s history using QRPEs

e.g., find a class for which a subclass is introduced in later version

(qwal
  graph start ?end
  [?class ?subclass]
  q=>*
  (qin-current
    (class ?class))
  q=>+
  (qin-current
    (class-subclass ?class ?subclass)
    (introduced ?subclass)))

e.g., find violations against test-driven development

A History Querying Tool and its Application to Detect Multi-version Refactorings
CSMR12 Reinout Stevens, Coen De Roover, Carlos Noguera, Viviane Jonckers
Reasoning over the Evolution of Source Code using QRPEs
WCRE11 Andy Kellens, Coen De Roover, Reinout Stevens, Carlos Noguera, Viviane Jonckers
More advanced declarative tools (1)

... QwalKeko: querying a program’s history using QRPEs

e.g., detect the pull up method refactoring

(qwal graph version1 version2
(qin-current
(ast :MethodDeclaration ?method)
(declaring-class ?method ?derived))

q=> ;;transition to next version

(qin-current
(method-moved ?method ?pulled)
(declaring-class ?pulled ?base)
(superclass ?base ?derived)))
More advanced declarative tools (2)

... example-driven bug detection

class ?name extends Component {
    public void acceptVisitor(?type ?v) {
        System.out.println(?string);
        ?v.?visitMethod(this);
    }
}

class OnlyLoggingLeaf extends Component {
    public void acceptVisitor(ComponentVisitor v) {
        System.out.println("Only logging.");
    }
}

class SillyLeaf extends OnlyLoggingLeaf {
    public void acceptVisitor(ComponentVisitor v) {
        super.acceptVisitor(v);
        ComponentVisitor temp = v;
        temp.visitSuperLogLeaf(this);
    }
}
More advanced declarative tools (2)

... example-driven bug detection
domain-specific extensions to unification ensure that implicit implementation variants unify

```java
class ?name extends Component {
    public void acceptVisitor(?type ?v) {
        System.out.println(?string);
        ?v.?visitMethod(this);
    }
}
```

```java
class MustAlias extends Component {
    public void acceptVisitor(ComponentVisitor v) {
        System.out.println("Hello");
        ComponentVisitor temp = v;
        temp.visitSuperLogLeaf(this);
    }
}
```

consults static analyses

<table>
<thead>
<tr>
<th>AST node</th>
<th>AST node</th>
<th>identical</th>
<th>1</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Qualified Type</th>
<th>Simple Type</th>
<th>denote same or co-variant return types</th>
<th>1</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Expression</th>
<th>Expression</th>
<th>in must-alias or may-alias relation</th>
<th>0.9 or 0.5</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Message Name</th>
<th>Method Name</th>
<th>message may invoke method according to dynamic or static receiver type</th>
<th>0.5 or 0.4</th>
</tr>
</thead>
</table>

likelihood of resulting in false positive, propagated by fuzzy logic cornerstone
In Practice: Detecting Lapsed Observer Bugs

![Diagram of Observer Pattern]

- **Subject**
  - Attach(Observer)
  - Detach(Observer)
  - Notify()
- **ConcreteSubject**
  - GetState()
  - SetState()
  - subjectState
- **Observer**
  - Update()
- **ConcreteObserver**
  - Update()
  - observerState

For all o in observers {
  o->Update()
}

observerState = subject->GetState()
we motivated each individual cornerstone of our approach—its specification language, its detection mechanism recognized implicit implementation variability.

Actual lapsed listeners could be identified through a subsequent dynamic analysis. The detection mechanism recognizes implicit implementation variability.

The lapsed listener pitfall at the instance-level: as instances are added to a class diagram and a sequence diagram in UML-like diagrams instead of code against reality. Using UML-like diagrams instead of code

Note that the depicted specification only detects possible conditions to the specification that implement heuristics. We have already started to explore this.

There are still many open questions related to our approach. Large corpus-based studies are needed to test these projections to reflect their projected likelihood of being a false positive.

Large corpus-based studies are needed to test these projections to reflect their projected likelihood of being a false positive.

Large corpus-based studies are needed to test these projections to reflect their projected likelihood of being a false positive.

Large corpus-based studies are needed to test these projections to reflect their projected likelihood of being a false positive.

Large corpus-based studies are needed to test these projections to reflect their projected likelihood of being a false positive.

Large corpus-based studies are needed to test these projections to reflect their projected likelihood of being a false positive.

Large corpus-based studies are needed to test these projections to reflect their projected likelihood of being a false positive.

Large corpus-based studies are needed to test these projections to reflect their projected likelihood of being a false positive.
we motivated each individual cornerstone of our approach and behavioral and its program representation. When fulfilled, these result in a pattern tool: its specification language, its detection mechanism desiderata for each dimension in the design of a pattern detection tool. As a result, we are able to detect most of the objects that are added to a struct.

For instance, the ranking of the detection results was intended to the missing instances would require exemplifying additional

V. F. µ-patterns and bug patterns.

For instance, the ranking of the detection results was intended to the missing instances would require exemplifying additional

V. F. µ-patterns and bug patterns.

For instance, the ranking of the detection results was intended to the missing instances would require exemplifying additional
More advanced declarative tools (n)

Query-by-UML
find implementations of exemplified architecture

Smart Annotations
monitor regularities about their own use

IntensiVE
monitor implementation with respect to design

jsRefact
tool-supported refactoring for JavaScript

Ekeko/X
workbench for automating systematic edits

code template for subject
change actions for matches
Ekeko/X: performing edits within a program

before

```java
public class BreakStatement extends Statement {
    @EntityProperty(value = SimpleName.class)
    private EntityIdentifier label;

    public EntityIdentifier getLabel() {
        return label;
    }

    public void setLabel(EntityIdentifier label) {
        this.label = label;
    }
}
```

after

```java
public class BreakStatement extends Statement {
    @EntityProperty(value = SimpleName.class)
    private EntityIdentifier<SimpleName> label;

    public EntityIdentifier<SimpleName> getLabel() {
        return label;
    }

    public void setLabel(EntityIdentifier<SimpleName> label) {
        this.label = label;
    }
}
```
Ekeko/X: performing edits within a program correctly

<table>
<thead>
<tr>
<th>Type</th>
<th>Eclipse JDT core</th>
<th>Eclipse SWT</th>
<th>Mozilla Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period of development</td>
<td>2001/06 ~ 2009/02</td>
<td>2001/05 ~ 2010/05</td>
<td>Several projects associated with Internet 1998/03 ~ 2008/05</td>
</tr>
<tr>
<td>Total revisions</td>
<td>17000 revisions</td>
<td>21530 revisions</td>
<td>200000 revisions</td>
</tr>
<tr>
<td># of bugs</td>
<td>1812</td>
<td>1256</td>
<td>11254</td>
</tr>
<tr>
<td># of Type I bugs</td>
<td>1405 (77.54%)</td>
<td>954 (75.96%)</td>
<td>7562 (67.19%)</td>
</tr>
<tr>
<td># of Type II bugs</td>
<td>407 (22.46%)</td>
<td>302 (24.04%)</td>
<td>3692 (32.81%)</td>
</tr>
</tbody>
</table>

Bugs requiring supplementary patches

Amount of supplements (~ missed occurrences)

Days before first supplement

[“An empirical study of supplementary bug fixes” Park et al. MSR 2012]
Automating Edits: using existing tool support

requires specifying

- subjects of the transformation (LHS)
- their state afterwards or change actions (RHS)

carefully ensuring

- no required changes are missed
- no unwarranted changes are applied
before

```java
public class BreakStatement extends Statement {
    @EntityProperty(value = SimpleName.class)
    private EntityIdentifier label;

    public EntityIdentifier getLabel() {
        return label;
    }

    public void setLabel(EntityIdentifier label) {
        this.label = label;
    }
}
```

after

```java
public class BreakStatement extends Statement {
    @EntityProperty(value = SimpleName.class)
    private EntityIdentifier<SimpleName> label;

    public EntityIdentifier<SimpleName> getLabel() {
        return label;
    }

    public void setLabel(EntityIdentifier<SimpleName> label) {
        this.label = label;
    }
```
;; Actual demo script
;; -----------------.

;; launches a logic query of which the solutions are all AST nodes of type :Annotation
(ekeko [?node]
  (ast :Annotation ?node))

;; logic queries can be embedded in functional expressions
(let [type :Annotation]
  (count
   (ekeko [?node] (ast type ?node))))

;; solutions are tuples of variable bindings (1-tuple of binding for ?ast) that satisfy the constraint
(map
  (comp class first) ; function composition, takes class of first element of 1-tuple
Repeating Changes: using Ekeko

and most state of the art tools

specifying changes requires significant expertise

\[
n : (x := v) \implies x := c
\]

if
\[
n \vdash A^\triangle (\neg \text{def}(v) \cup \text{def}(v) \land \text{stmt}(v := c))
\]

conlit(c)

often requires multiple iterations

no required changes are missed

no unwarranted changes are applied
1/ template-based transformation specifications
2/ embedded directives for generalizing and refining template matching & instantiation
3/ future work: recommender system for directives

```
(ekeko/x
  @EntityProperty(value=?annoType.class)
  private EntityIdentifier ...;
  =>
  EntityIdentifier<?annoType>)
```

**template on LHS:** matches = edit locations

**template on RHS:** instantiated for each LHS match

**BUT:** LHS does not yet match fields with other modifiers
RHS does not have a destination for the generated code
1/ template-based transformation specifications
2/ embedded directives for generalizing and refining template matching & instantiation
3/ future work: recommender system for directives

```java
(ekeko/x
    [@EntityProperty(value=?annoType.class)
    private]@[match|set]
EntityIdentifier@[(equals ?fieldType)] ...;
=>
EntityIdentifier<?annoType>@[(replace ?fieldType)]
```

matching directive

rewriting directive
1/ template-based transformation specifications
2/ embedded directives for generalizing and refining template matching & instantiation
3/ future work: recommender system for directives
1/ template-based transformation specifications

Embedded directives control matching & instantiation

Recommender system for suitable directives
1/ template-based transformation specifications

2/ embedded directives for generalizing and refining template matching & instantiation

contribution: syntactic, structural, control flow, data flow characteristics

3/ future work: recommender system for directives
Applications: beyond automating edits

domain-specific refactorings
refactorings to eliminate or maintain clones

program renovation
convert anonymous classes to Java 8 lambda expressions

API migration
migrate an application from using SWING to using SWT

domain-specific optimizations

List listUsers := [ ]; int i, j = 0;
List users := Query(SELECT * FROM users);
List roles = Query(SELECT * FROM roles);
while (i < users.size()) {
  while (j < roles.size()) {
    if (users[i].roleld == roles[j].roleld)
      listUsers := append(listUsers, users[i]);
    ++j;
  }
  ++i;
}

List<User> getRoleUser () {
  List<User> listUsers = db.executeQuery(
    "SELECT u
    FROM users u, roles r
    WHERE u.roleld == r.roleld
    ORDER BY u.roleld, r.roleld";
  return listUsers; }

[Cheung et al., PLDI13]