Exploring a Method to Detect Behaviour-Preserving Evolution Using Graph Transformation

Javier Pérez, Yania Crespo
`{jperez,yania}@infor.uva.es`

Universidad de Valladolid

Third International ERCIM Symposium on Software Evolution
(co-located with ICSM 2007)

Introduction: Context

- Refactorings are commonly integrated into development environments and are extensively used.
- Finding and understanding refactorings is important to document and to understand a system's evolution.
- It will be useful to determine automatically when software evolution has been behaviour-preserving.
  - to verify a redesign process
  - to verify a handmade refactoring
  - to find and characterise stages of a system's evolution
  - ...

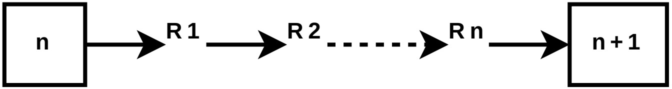
Introduction: Goals

- To detect whether two versions of a software system are functionally equivalent,
- by checking whether the evolution process between this two versions can be formulated by a refactoring sequence

$$\boxed{\mathbf{n}} \qquad \simeq \qquad \boxed{\mathbf{n+1}}$$

Introduction: Goals

- To detect whether two versions of a software system are functionally equivalent,
- by checking whether the evolution process between this two versions can be formulated by a refactoring sequence



Introduction: Approach

- We are exploring a method which:
  - uses a graph representation format for Java programs and Java refactorings,
  - models the problem as a state space search,
  - searches sequences of ONLY refactorings,
  - uses refactorings' pre and postconditions to guide the search.

Introduction: Approach

- We are exploring a method which:
  - uses a graph representation format for Java programs and Java refactorings,
  - models the problem as a state space search,
  - searches sequences of ONLY refactorings,
  - uses refactorings’ pre and postconditions to guide the search.

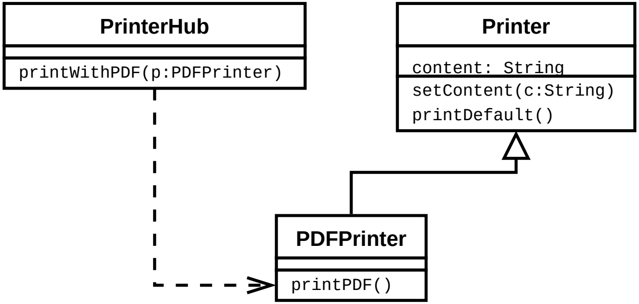
Introduction: Approach

- We are exploring a method which:
  - uses a graph representation format for Java programs and Java refactorings,
  - models the problem as a state space search,
  - searches sequences of ONLY refactorings,
  - uses refactorings' pre and postconditions to guide the search.

Introduction: Approach

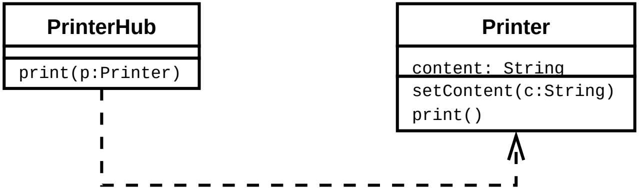
- We are exploring a method which:
  - uses a graph representation format for Java programs and Java refactorings,
  - models the problem as a state space search,
  - searches sequences of ONLY refactorings,
  - uses refactorings' pre and postconditions to guide the search.

Example: Simulation of a Printing System



Different printers for different document types and a printer hub to connect all the printers. More printers will be added when needed.

Example: Refactored Printing System



The system administrator noticed that only pdf documents were sent.
Rashly modification to simplify the inheritance hierarchy.

Problem!

- A new system administrator arrives. Finds two versions of the system, and no documentation about the changes performed between them.
- **Problems:**
  - documenting the changes performed to the old system
  - is the new system functionally equivalent to the old one?
- We know that a sequence exists (done manually).

Refactoring Sequence Applied

1. ***removeMethod*:**
   printing.Printer.printDefault()
2. ***pullUpMethod*:**
   printing.PDFPrinter.printPDF() $\Longrightarrow$ *printing.Printer.printPDF()*
3. ***renameMethod*:**
   printing.Printer.printPDF() $\Longrightarrow$ *printing.Printer.print()*
4. ***renameMethod*:**
   printing.PrinterHub.printWithPDF() $\Longrightarrow$ *printing.PrinterHub.print()*
5. ***useSuperType*:**
   printing.Printer.print(PDFPrinter p) $\Longrightarrow$
   printing.Printer.print(Printer p)
6. ***removeClass*:**
   printing.PDFPrinter

Refactoring Sequence Applied

1. ***removeMethod*:** *printing.Printer.printDefault()*
2. ***pullUpMethod*:** *printing.PDFPrinter.printPDF()* $\Longrightarrow$ *printing.Printer.printPDF()*
3. ***renameMethod*:** *printing.Printer.printPDF()* $\Longrightarrow$ *printing.Printer.print()*
4. ***renameMethod*:** *printing.PrinterHub.printWithPDF()* $\Longrightarrow$ *printing.PrinterHub.print()*
5. ***useSuperType*:** *printing.Printer.print(PDFPrinter p)* $\Longrightarrow$ *printing.Printer.print(Printer p)*
6. ***removeClass*:** *printing.PDFPrinter*

Refactoring Sequence Applied

1. ***removeMethod*:**
   printing.Printer.printDefault()
2. ***pullUpMethod*:**
   printing.PDFPrinter.printPDF() $\Longrightarrow$ *printing.Printer.printPDF()*
3. ***renameMethod*:**
   printing.Printer.printPDF() $\Longrightarrow$ *printing.Printer.print()*
4. ***renameMethod*:**
   printing.PrinterHub.printWithPDF() $\Longrightarrow$ *printing.PrinterHub.print()*
5. ***useSuperType*:**
   printing.Printer.print(PDFPrinter p) $\Longrightarrow$
   printing.Printer.print(Printer p)
6. ***removeClass*:**
   printing.PDFPrinter

Refactoring Sequence Applied

1. ***removeMethod*:**
   printing.Printer.printDefault()
2. ***pullUpMethod*:**
   printing.PDFPrinter.printPDF() $\Longrightarrow$ *printing.Printer.printPDF()*
3. ***renameMethod*:**
   printing.Printer.printPDF() $\Longrightarrow$ *printing.Printer.print()*
4. ***renameMethod*:**
   printing.PrinterHub.printWithPDF() $\Longrightarrow$ *printing.PrinterHub.print()*
5. ***useSuperType*:**
   printing.Printer.print(PDFPrinter p) $\Longrightarrow$
   printing.Printer.print(Printer p)
6. ***removeClass*:**
   printing.PDFPrinter

Refactoring Sequence Applied

1. ***removeMethod*:**
   printing.Printer.printDefault()
2. ***pullUpMethod*:**
   printing.PDFPrinter.printPDF() $\Longrightarrow$ *printing.Printer.printPDF()*
3. ***renameMethod*:**
   printing.Printer.printPDF() $\Longrightarrow$ *printing.Printer.print()*
4. ***renameMethod*:**
   printing.PrinterHub.printWithPDF() $\Longrightarrow$ *printing.PrinterHub.print()*
5. ***useSuperType*:**
   printing.Printer.print(PDFPrinter p) $\Longrightarrow$
   printing.Printer.print(Printer p)
6. ***removeClass*:**
   printing.PDFPrinter

Refactoring Sequence Applied

1. ***removeMethod*:**
   printing.Printer.printDefault()
2. ***pullUpMethod*:**
   printing.PDFPrinter.printPDF() $\Longrightarrow$ *printing.Printer.printPDF()*
3. ***renameMethod*:**
   printing.Printer.printPDF() $\Longrightarrow$ *printing.Printer.print()*
4. ***renameMethod*:**
   printing.PrinterHub.printWithPDF() $\Longrightarrow$ *printing.PrinterHub.print()*
5. ***useSuperType*:**
   printing.Printer.print(PDFPrinter p) $\Longrightarrow$
   printing.Printer.print(Printer p)
6. ***removeClass*:**
   printing.PDFPrinter

Refactorings and Graph Transformation

- We use Graph Transformation as a formal representation for refactorings and OO software
  - GT deals with structure representation and modification
  - refactorings are structural modifications
- We use the work of Mens et al. *“Formalising Refactorings with Graph Transformations”* as our basis, to represent:
  - programs as graphs
  - refactorings as graph transformation rules
- We have made a small extension to this format to represent simple Java programs.

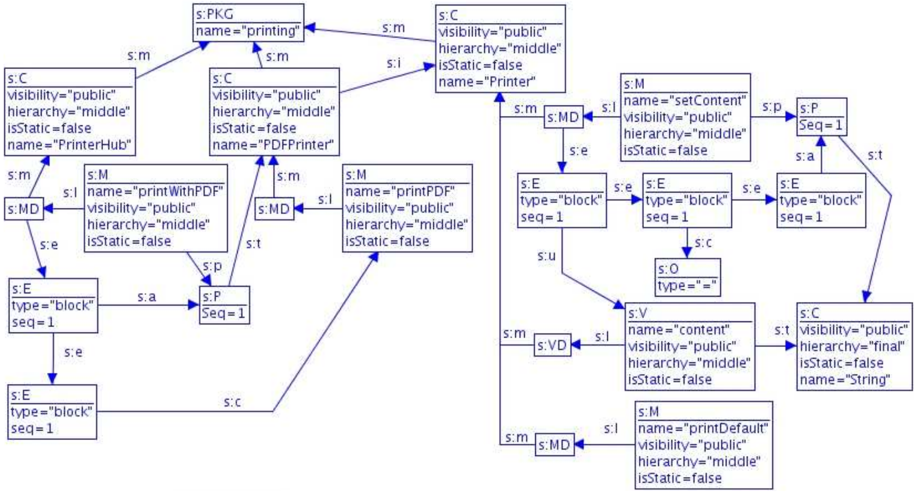
Refactorings and Graph Transformation

- We use Graph Transformation as a formal representation for refactorings and OO software
  - GT deals with structure representation and modification
  - refactorings are structural modifications
- We use the work of Mens et al. *“Formalising Refactorings with Graph Transformations”* as our basis, to represent:
  - programs as graphs
  - refactorings as graph transformation rules
- We have made a small extension to this format to represent simple Java programs.

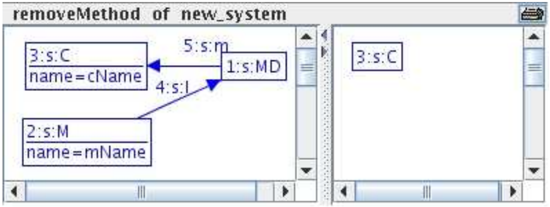
Original Printing System

```
//-----------------------PrinterHub.java
public class PrinterHub {
  public void printWithPDF(PDFPrinter p){
    p.printPDF();
  }
}
//-----------------------PDFPrinter.java
public class PDFPrinter extends Printer{
  public void printPDF(){
    // body of printPDF method
  }
}
//--------------------------Printer.java
public class Printer {
        public String content;
        public void setContent(String c){
                this.content = c;
        }
        public void printDefault(){
          // body of printDefault method
        }
}
```

Original Printing System Graph



Refactorings as Graph Transformation Rules



- **Left-hand side:** Rule precondition.
  - Can be used to express refactorings' pre and postconditions.
- **Rigth-hand side:** Transformation.

Modeling the problem

- We address the problem as a state space search problem:
  - **Original/Old system** $\simeq$ start state.
  - **Refactoring operations** $\simeq$ state changing operations, edges.
  - **Refactored/New system** $\simeq$ goal state.
  - **Does a refactoring sequence exist?** $\simeq$ reachability problem.
  - **Refactoring sequence** $\simeq$ path from the start state to the goal state.
- We apply a graph parsing algorithm to perform depth-first search
- **Main problem:** size of the state space (finite?)
  - With refactoring descriptions expressed in terms of preconditions, transformations and postconditions,
  - preconditions and postconditions can guide the search,
  - we can reduce the size of the state space.

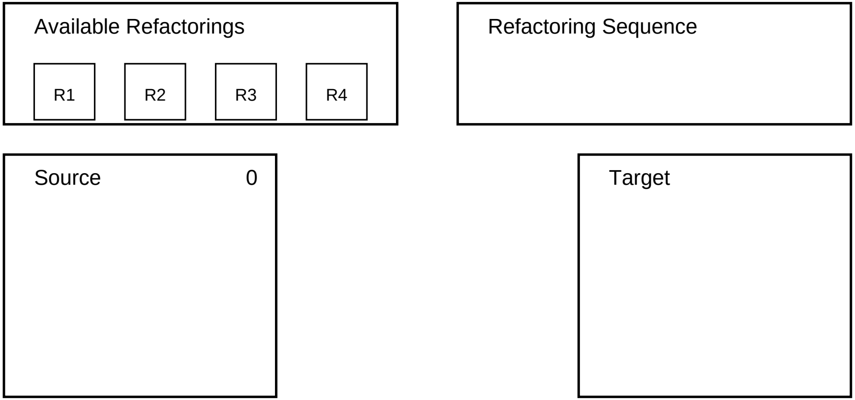
Modeling the problem

- We address the problem as a state space search problem:
  - **Original/Old system** $\simeq$ start state.
  - **Refactoring operations** $\simeq$ state changing operations, edges.
  - **Refactored/New system** $\simeq$ goal state.
  - **Does a refactoring sequence exist?** $\simeq$ reachability problem.
  - **Refactoring sequence** $\simeq$ path from the start state to the goal state.
- We apply a graph parsing algorithm to perform depth-first search
- **Main problem:** size of the state space (finite?)
  - With refactoring descriptions expressed in terms of preconditions, transformations and postconditions,
  - preconditions and postconditions can guide the search,
  - we can reduce the size of the state space.

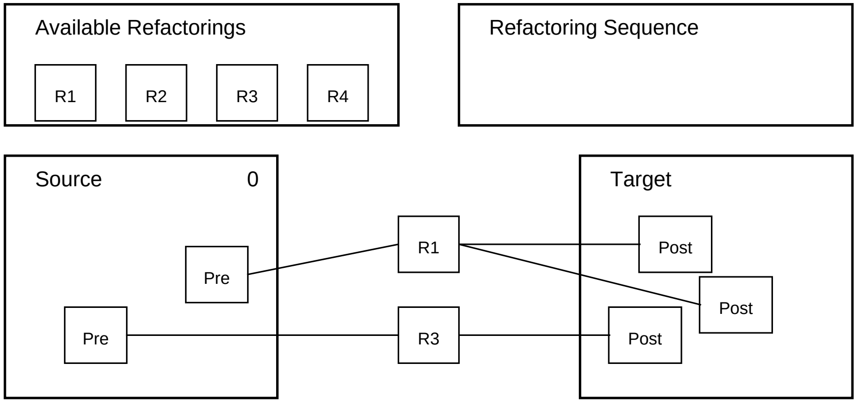
Modeling the problem

- We address the problem as a state space search problem:
    - **Original/Old system** $\simeq$ start state.
    - **Refactoring operations** $\simeq$ state changing operations, edges.
    - **Refactored/New system** $\simeq$ goal state.
    - **Does a refactoring sequence exist?** $\simeq$ reachability problem.
    - **Refactoring sequence** $\simeq$ path from the start state to the goal state.
- We apply a graph parsing algorithm to perform depth-first search
- **Main problem:** size of the state space (finite?)
    - With refactoring descriptions expressed in terms of preconditions, transformations and postconditions,
    - preconditions and postconditions can guide the search,
    - we can reduce the size of the state space.

Algorithm

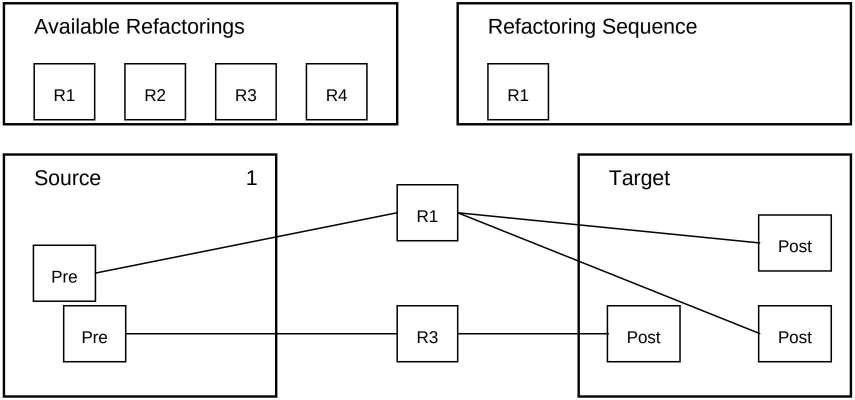

Available Refactorings
R1
R2
R3
R4
Refactoring Sequence
Source
0
Target

Algorithm



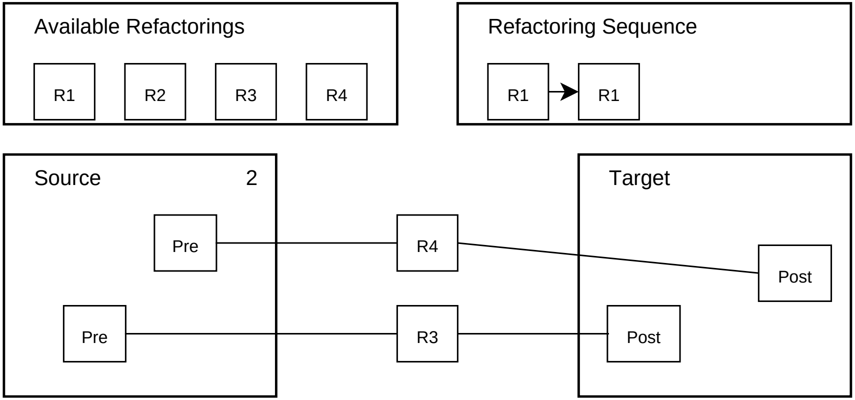
- Looks for refactoring preconditions in the start graph.
- Looks for refactoring postconditions in the goal graph.

Algorithm



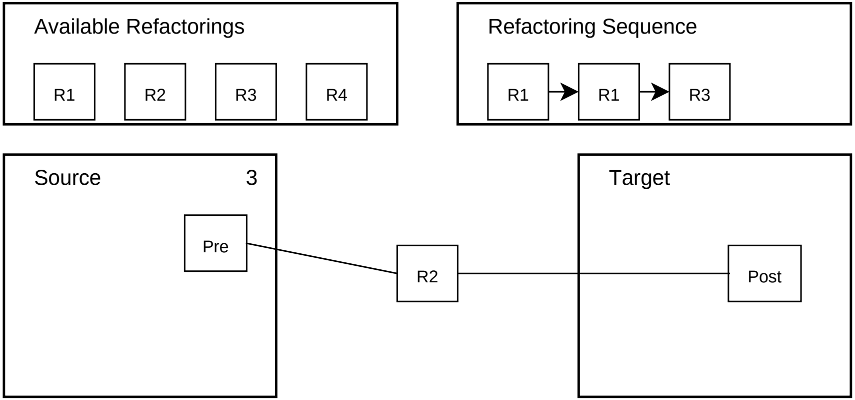
- Iteratively selects candidate refactorings
- Transforms the current graph with them

Algorithm

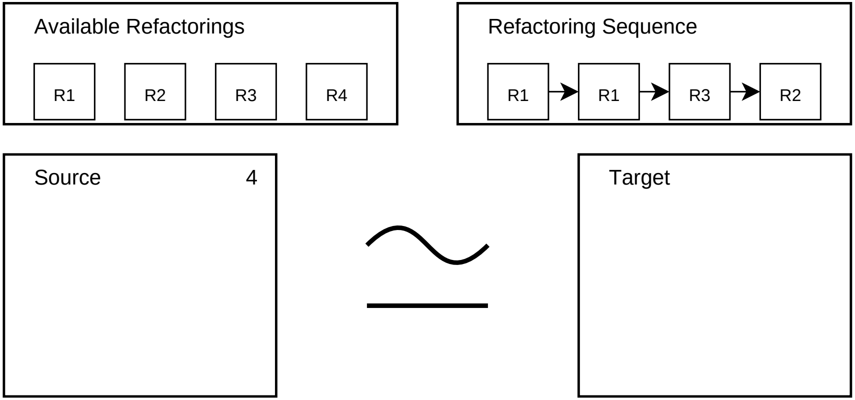


- Iteratively selects candidate refactorings
- Transforms the current graph with them

Algorithm


Available Refactorings
R1
R2
R3
R4
Refactoring Sequence
R1
R1
R3
Source
3
Pre
R2
Target
Post

Algorithm



- Success: current graph isomorphic to the goal graph,
- Fail: No more refactorings can be executed, current and goal states are not isomorphic.

Implementation in AGG

- Easy to use graph transformation tool
- AGG allows rapid prototyping of GT systems.
- It supports graph parsing, which can be used to perform the search:
    - The AGG parser randomly applies rules to the start graph
    - until it is isomorphic to the goal graph,
    - or no more rules are available,
    - and backtracking is no longer possible.
- AGG allows to “exercise” our approach easily.
- It present expressiveness and efficiency limitations.

Implementation in AGG

- Easy to use graph transformation tool
- AGG allows rapid prototyping of GT systems.
- It supports graph parsing, which can be used to perform the search:
    - The AGG parser randomly applies rules to the start graph
    - until it is isomorphic to the goal graph,
    - or no more rules are available,
    - and backtracking is no longer possible.
- AGG allows to “exercise” our approach easily.
- It present expressiveness and efficiency limitations.

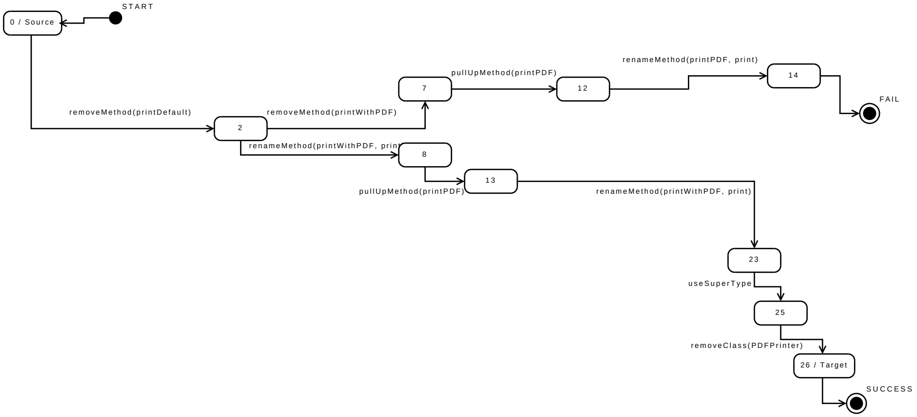
Running the Example

- Set of rules to search:
  - pullUpMethod, renameMethod, removeMethod, removeClass, removeInterface and useSuperType
- Each iteration, among candidate rules:
  - AGG selects randomly one to apply it.
  - AGG backtracks when needed and possible.
- Output from the AGG parser's debugging information:
  - rules applied
  - when backtracking occurs
  - intermediate graphs
  - ...
- parsing takes about 2 seconds

Running the Example

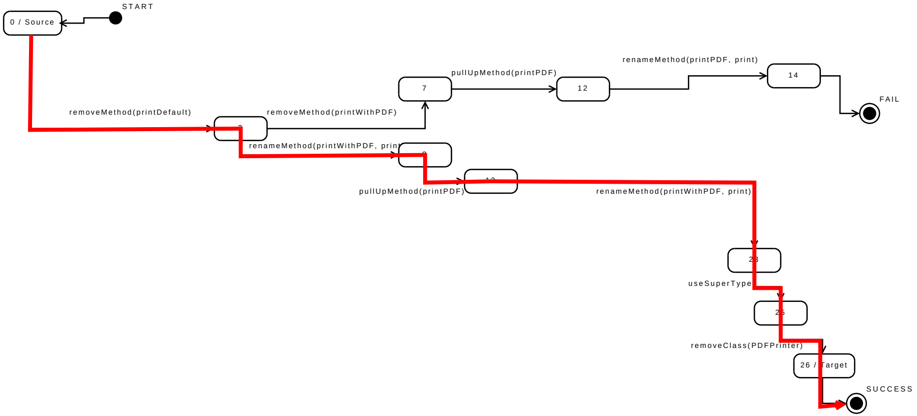
- Set of rules to search:
    - pullUpMethod, renameMethod, removeMethod, removeClass, removeInterface and useSuperType
- Each iteration, among candidate rules:
    - AGG selects randomly one to apply it.
    - AGG backtracks when needed and possible.
- Output from the AGG parser's debugging information:
    - rules applied
    - when backtracking occurs
    - intermediate graphs
    - ...
- parsing takes about 2 seconds

State Space, Derivation Graph


START
0 / Source
removeMethod(printDefault)
2
removeMethod(printWithPDF)
7
pullUpMethod(printPDF)
12
renameMethod(printPDF, print)
14
FAIL
renameMethod(printWithPDF, print)
8
pullUpMethod(printPDF)
13
renameMethod(printWithPDF, print)
23
useSuperType
25
removeClass(PDFPrinter)
26 / Target
SUCCESS

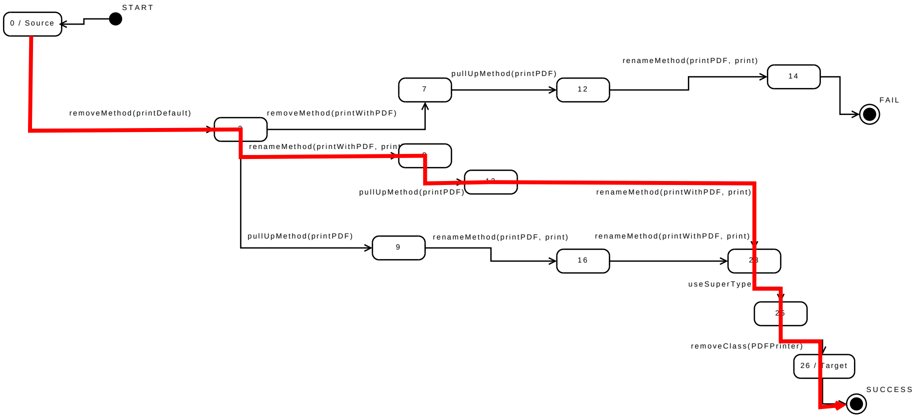
For this experiment, the parsing terminates and finds a sequence.

State Space, Derivation Graph


START
0 / Source
removeMethod(printDefault)
removeMethod(printWithPDF)
7
pullUpMethod(printPDF)
12
renameMethod(printPDF, print)
14
FAIL
renameMethod(printWithPDF, print)
pullUpMethod(printPDF)
renameMethod(printWithPDF, print)
23
useSuperType
25
removeClass(PDFPrinter)
26 / Target
SUCCESS

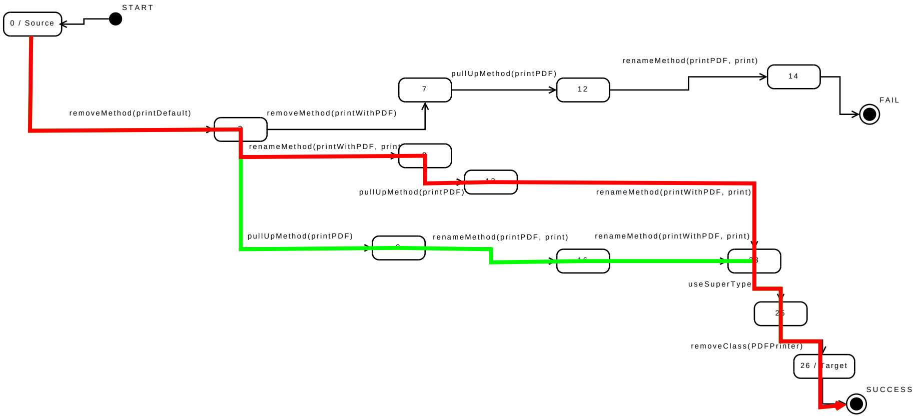
For this experiment, the parsing terminates and finds a sequence.

State Space, Derivation Graph


START
0 / Source
removeMethod(printDefault)
removeMethod(printWithPDF)
7
pullUpMethod(printPDF)
12
renameMethod(printPDF, print)
14
FAIL
renameMethod(printWithPDF, print)
pullUpMethod(printPDF)
renameMethod(printWithPDF, print)
pullUpMethod(printPDF)
9
renameMethod(printPDF, print)
16
renameMethod(printWithPDF, print)
23
useSuperType
25
removeClass(PDFPrinter)
26 / Target
SUCCESS

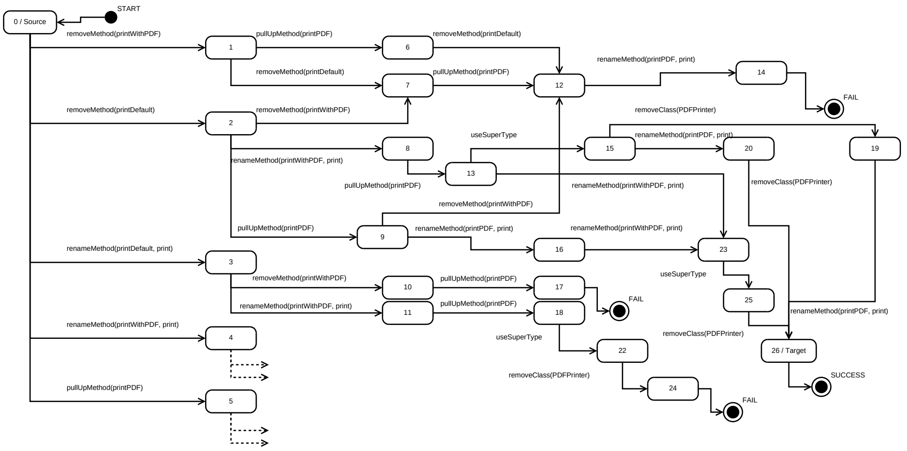
- The first sequence found differs from the one found manually

State Space, Derivation Graph


START
0 / Source
removeMethod(printDefault)
removeMethod(printWithPDF)
7
pullUpMethod(printPDF)
12
renameMethod(printPDF, print)
14
FAIL
renameMethod(printWithPDF, print)
pullUpMethod(printPDF)
renameMethod(printWithPDF, print)
pullUpMethod(printPDF)
renameMethod(printPDF, print)
renameMethod(printWithPDF, print)
useSuperType
removeClass(PDFPrinter)
26 / Target
SUCCESS

- The first sequence found differs from the one found manually

State Space, Derivation Graph


START
0 / Source
removeMethod(printWithPDF)
pullUpMethod(printPDF)
removeMethod(printDefault)
renameMethod(printPDF, print)
FAIL
removeClass(PDFPrinter)
useSuperType
renameMethod(printWithPDF, print)
renameMethod(printDefault, print)
SUCCESS
26 / Target
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
22
23
24
25

- We can obtain the whole state space. In this case, it is finite.

Our results

- There are not many works dealing with finding refactorings.
- These efforts focus in mining refactorings mixed with other changes.
- We focus on the detection of behaviour-preserving evolution. Changes are only refactorings.

- We can deal with multiple refactoring changes applied to the same piece of code.
- We can deal with renamings.
- The structural representation can be as detailed as needed to support refactorings at any abstraction level.

- We have explored the possibilities of our approach.
- Many ways of improving it to solve the open problems.

Our results

- There are not many works dealing with finding refactorings.
- These efforts focus in mining refactorings mixed with other changes.
- We focus on the detection of behaviour-preserving evolution. Changes are only refactorings.

- We can deal with multiple refactoring changes applied to the same piece of code.
- We can deal with renamings.
- The structural representation can be as detailed as needed to support refactorings at any abstraction level.

- We have explored the possibilities of our approach.
- Many ways of improving it to solve the open problems.

Our results

- There are not many works dealing with finding refactorings.
- These efforts focus in mining refactorings mixed with other changes.
- We focus on the detection of behaviour-preserving evolution. Changes are only refactorings.

- We can deal with multiple refactoring changes applied to the same piece of code.
- We can deal with renamings.
- The structural representation can be as detailed as needed to support refactorings at any abstraction level.

- We have explored the possibilities of our approach.
- Many ways of improving it to solve the open problems.

Problems and Limitations: Termination

- **Problem:**
  - Our searching algorithm is only partially correct.
  - If the state space is not finite the termination can not be guaranteed.
- **Solutions:**
  - Use of refactorings' pre and postconditions
  - Formulate the searching rules to limit the search space size.
  - Store states to not check the same state twice.
  - More heuristics.

Problems and Limitations: Termination

- **Problem:**
  - Our searching algorithm is only partially correct.
  - If the state space is not finite the termination can not be guaranteed.
- **Solutions:**
  - Use of refactorings' pre and postconditions
  - Formulate the searching rules to limit the search space size.
  - Store states to not check the same state twice.
  - More heuristics.

Problems and Limitations: expressiveness

- **Problem:**
  - We have not implemented “real” refactoring operations.
  - AGG lacks some key features needed, path expressions.
  - Representing context:
    - Limitation to a single, finite context.
    - We need to specify a set of contexts.
- **Solution:**
  - Test more GT tools (PROGRES, GROOVE, . . .).

Problems and Limitations: expressiveness

- **Problem:**
    - We have not implemented “real” refactoring operations.
    - AGG lacks some key features needed, path expressions.
    - Representing context:
        - Limitation to a single, finite context.
        - We need to specify a set of contexts.
- **Solution:**
    - Test more GT tools (PROGRES, GROOVE, . . .).

Problems and Limitations: Complex Refactorings

- **Problem:**
  - Difficult to represent rules for refactorings which take an undetermined number of steps.
  - Lack of a full transformation control in AGG,
- **Solutions:**
  - Last versions of AGG implement a better execution control.
  - Program the rule control and use AGG as a backend rule execution engine.
  - Test more GT tools (PROGRES, GROOVE, . . .).

Problems and Limitations: Complex Refactorings

- **Problem:**
  - Difficult to represent rules for refactorings which take an undetermined number of steps.
  - Lack of a full transformation control in AGG,
- **Solutions:**
  - Last versions of AGG implement a better execution control.
  - Program the rule control and use AGG as a backend rule execution engine.
  - Test more GT tools (PROGRES, GROOVE, . . .).

Future Work

- **Analysis of the state space:** Can we formulate the refactoring searching rules to restrict the search space to a finite state space?
- **Searching rule catalog:** Improving rules with features in the newest AGG's versions. Implementing rules to search more refactorings.
- **Test other GT tools:** To improve efficiency, expressiveness, . . .
- **Full Java model**: Use another metamodel which can represent full Java programs.
- **Scalability:** Measuring the scalability and reliability of our technique over industrial-size systems.
- **Tool:** Eclipse plugin front-end to translate code to graphs, to launch AGG and to show up the refactoring sequence in a more convenient way.

Exploring a Method to Detect Behaviour-Preserving Evolution Using Graph Transformation

Javier Pérez, Yania Crespo
`{jperez,yania}@infor.uva.es`

Universidad de Valladolid

Third International ERCIM Symposium on Software Evolution
(co-located with ICSM 2007)