VirginiaTech
Invent the Future

Fine-grained and Accurate Source Code Differencing: GumTree

ASE 2014. (citation#;46)

Kijin An, CS6704

VirginiaTech
Invent the Future

Introduction

- Unix diff tool takes as input two versions of a source code file and performs the **Myers algorithm** [24]

- Limitations of diff like tools
    - First, it only computes **additions** and **deletions** and does not consider actions such as **update** and **move**
    - Second, it works at a granularity (**the text line**) that is both coarse grain and not aligned with the source code structure: the abstract syntax tree (AST)

Examples

```java
public class Test {
        public String foo(int i) {
                if (i == 0) return "Foo!";

        }
}
```

```java
public class Test {
        private String foo(int i) {
                if (i == 0) return "Bar";
            else if (i == -1) return "Foo!";
        }
}
```

Diff tools over Java codes

2 removals
3 additions

Test.java (source)

```java
public class Test {
        public String foo(int i) {
                if (i == 0) return "Foo!";
        }
}
```

Test.java (destination)

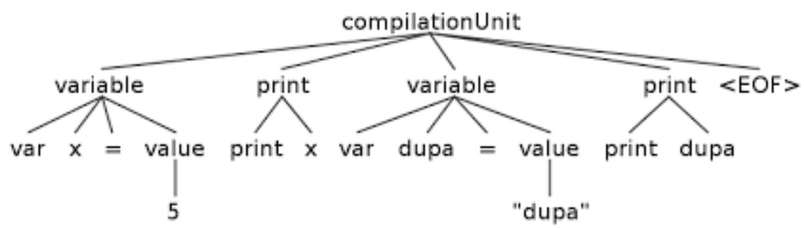
```java
public class Test {
        private String foo(int i) {
                if (i == 0) return "Bar";
                else if (i == -1) return "Foo!";

        }
}
```

AST differencing tools over Java codes

goals: to find a sequence "***edit actions"*** in AST levels

Edit Actions in AST Differencing

- Abstract Syntax Tree (AST)
  - Labeled ordered rooted tree: labels and values
    - Label: the name of their production rule in the grammar



- Four AST edit actions in AST Differencing
  - ***updateValue***(t, v_n)
  - ***add***(t, t_p, i, l, v)
  - ***delete***(t)
  - ***move***(t, t_p, i)

Edit Actions in AST Differencing

- Four AST edit actions
  - ***updateValue***(t, v_n) :replaces the old value of t by the new value v_n
  - ***add***(t, t_p, i, l, v) :adds a new node t in the AST. If t_p is not null and *i* is specified then t is the i^{th} child of t_p. Finally, l is the label of t and v is the value of t
  - ***delete***(t) :removes a leaf node of the AST
  - ***move***(t, t_p, i) :moves a node t and make it the i^{th} child of t_p. Note that all children of t are moved as well, therefore this actions moves a whole subtree

VirginiaTech
Invent the Future

Edit Actions in AST Differencing

- Edit Actions Examples

Test.java (source)

```
public class Test {
        public String foo(int i) {
                if (i == 0) return "Foo!";
        }
}
```

Test.java (destination)

```
public class Test {
        private String foo(int i) {
                if (i == 0) return "Bar";
                else if (i == -1) return "Foo!";

        }
}
```

- Which one is
  - ***updateValue***?
  - ***add***?
  - ***move***?
  - ***remove***?

Overall GumTree Algorithm

- AST differencing algorithms work in two steps:
  - 1.Establishing **mappings** of **src** and **dst**
    - » only explain in detail how we look for the mappings between two ASTs
  - 2.then deducing an **edit script**
    - » an optimal and quadratic algorithm has already been developed by the Chawathe et al. [6]

GumTree Algorithm for mapping

- GumTree **Mapping** Algorithms
  - A greedy **top-down** algorithm
  - A **bottom-up** algorithm
  - **recovery mappings**

VirginiaTech
Invent the Future

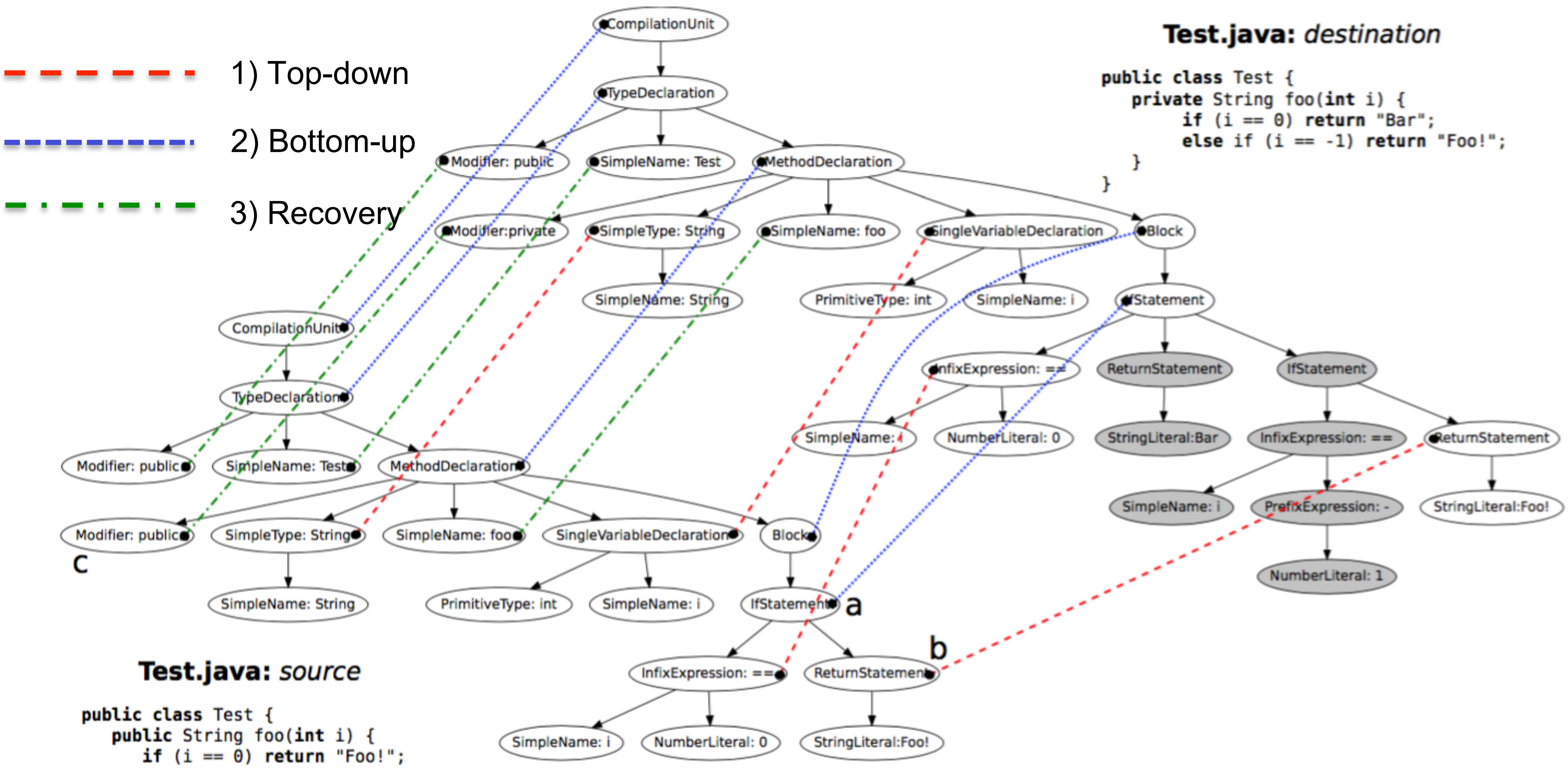
GumTree Algorithm for mapping

- **A greedy top-down algorithm:** anchors mappings
- **A bottom-up algorithm:** called a container mapping
- **recovery mappings:** to search for additional mappings among their descendants

- First they search for the biggest unmodified pieces of code
- Then they deduce which container of code can be mapped together
- Finally they look at precise differences in what is leftover in each container

VirginiaTech
Invent the Future

AST Differencing



Test.java: *destination*

```
public class Test {
   private String foo(int i) {
        if (i == 0) return "Bar";
        else if (i == -1) return "Foo!";
   }
}
```

Test.java: *source*

```
public class Test {
   public String foo(int i) {
      if (i == 0) return "Foo!";
   }
}
```

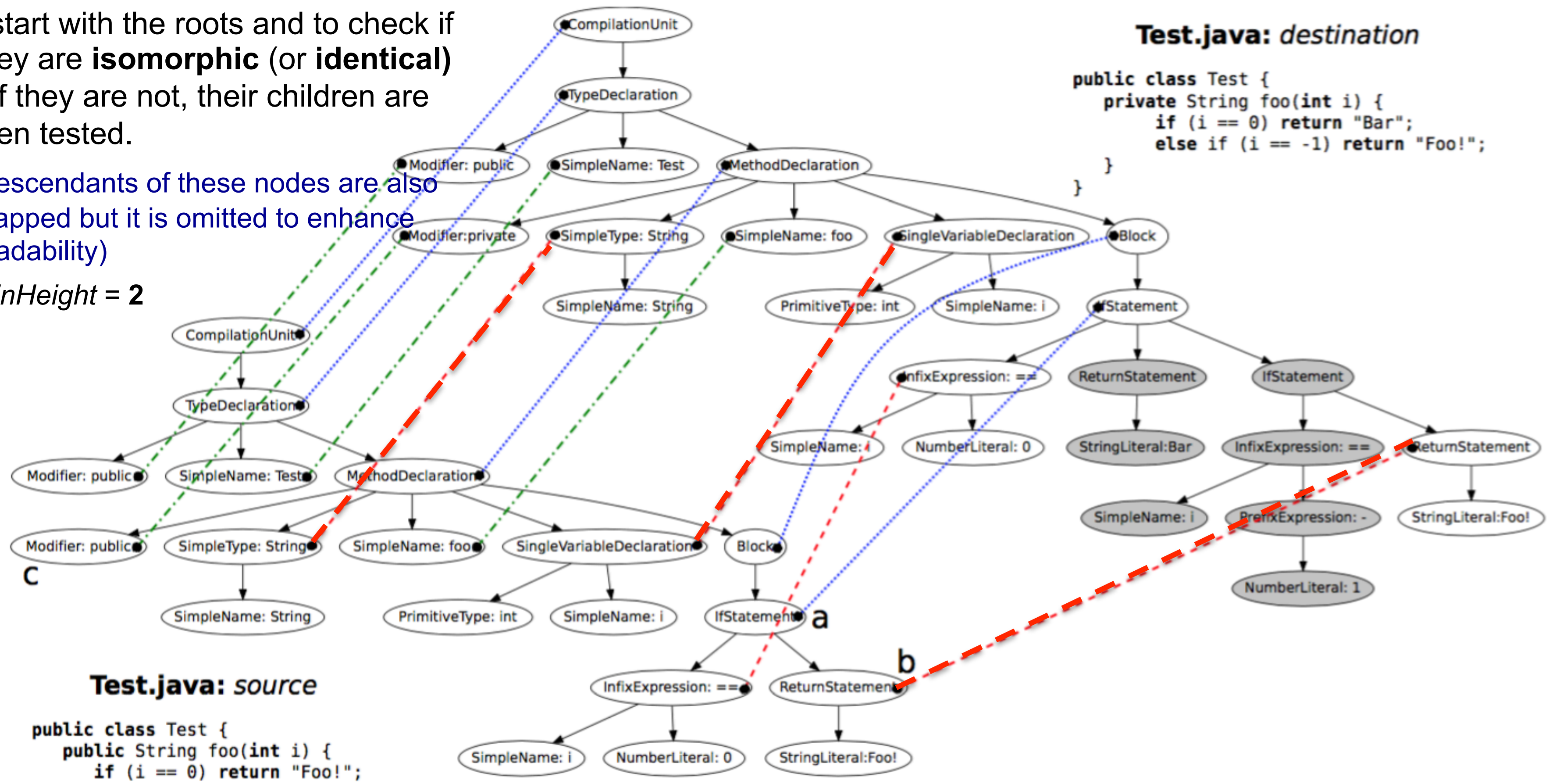
AST Differencing mapping with top-down

Top-down

- start with the roots and to check if they are **isomorphic** (or **identical)**
- If they are not, their children are then tested.

(descendants of these nodes are also mapped but it is omitted to enhance readability)

minHeight = **2**



Test.java: *destination*

```
public class Test {
    private String foo(int i) {
        if (i == 0) return "Bar";
        else if (i == -1) return "Foo!";
    }
}
```

Test.java: *source*

```
public class Test {
  public String foo(int i) {
    if (i == 0) return "Foo!";
  }
}
```

AST Differencing mapping with top-down

Algorithm 1: The algorithm of the top-down phase.

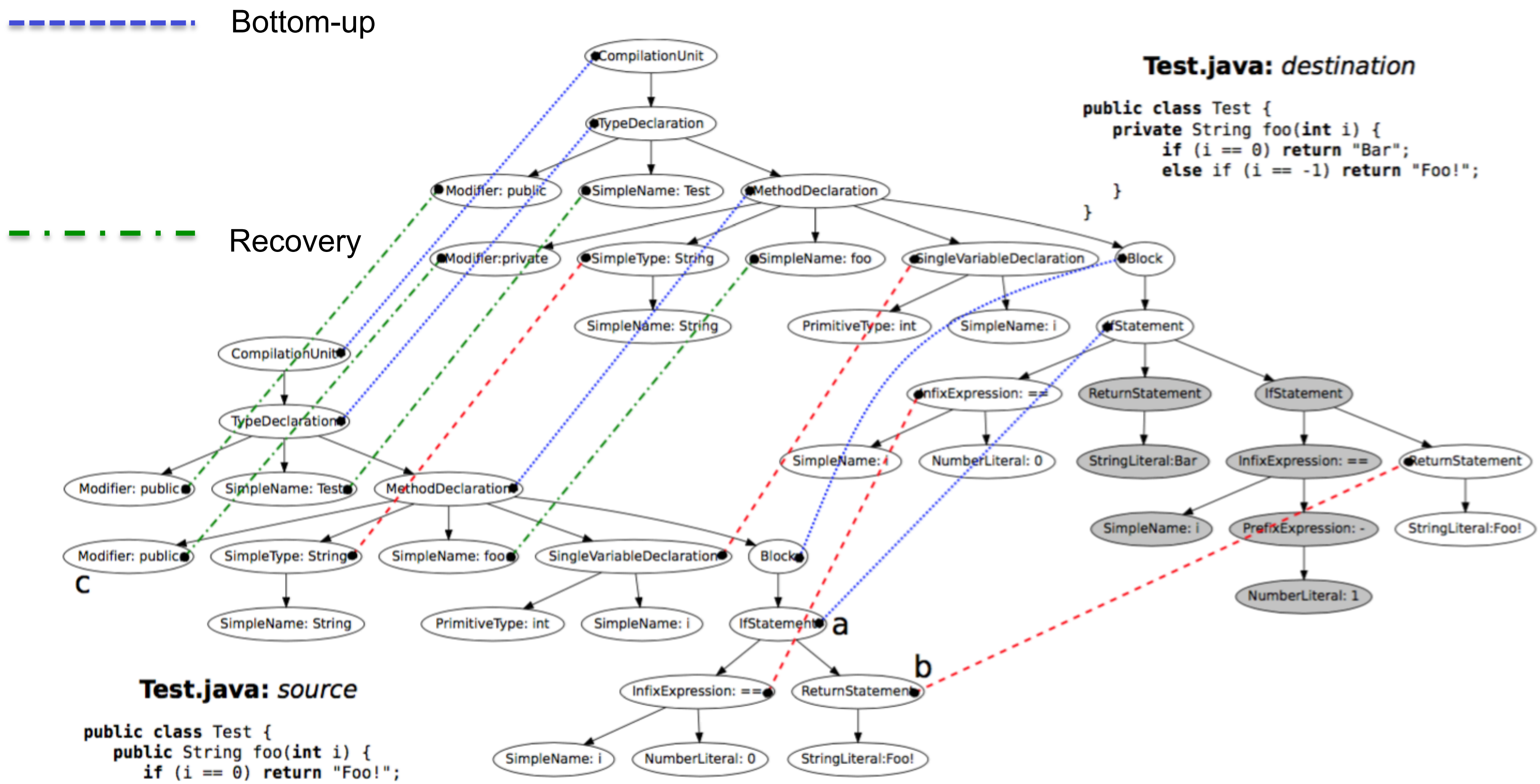
Data: A source tree T_1 and a destination tree T_2, a minimum height $minHeight$, two empty height-indexed priority lists L_1 and L_2, an empty list $\mathcal{A}$ of candidate mappings, and an empty set of mappings $\mathcal{M}$

Result: The set of mappings $\mathcal{M}$

$push(root(T_1), L_1)$;
$push(root(T_2), L_2)$;
while $min(peekMax(L_1), peekMax(L_2)) > minHeight$ **do**
  if $peekMax(L_1) \neq peekMax(L_2)$ **then**
    if $peekMax(L_1) > peekMax(L_2)$ **then**
      foreach $t \in pop(L_1)$ **do** $open(t, L_1)$;
    else
      foreach $t \in pop(L_2)$ **do** $open(t, L_2)$;
  else
    $H_1 \leftarrow pop(L_1)$;
    $H_2 \leftarrow pop(L_2)$;
    foreach $(t_1, t_2) \in H_1 \times H_2$ **do**
      if isomorphic(t_1, t_2) **then**
        if $\exists t_x \in T_2 \mid$ isomorphic$(t_1, t_x) \wedge t_x \neq t_2$ **or** $\exists t_x \in T_1 \mid$ isomorphic$(t_x, t_2) \wedge t_x \neq t_1$ **then**
          $add(\mathcal{A}, (t_1, t_2))$;
        else
          add all pairs of isomorphic nodes of $s(t_1)$ and $s(t_2)$ to $\mathcal{M}$;
    foreach $t_1 \in H_1 \mid (t_1, t_x) \notin \mathcal{A} \cup \mathcal{M}$ **do** $open(t_1, L_1)$;
    foreach $t_2 \in H_2 \mid (t_x, t_2) \notin \mathcal{A} \cup \mathcal{M}$ **do** $open(t_2, L_2)$;
sort $(t_1, t_2) \in \mathcal{A}$ using dice$(parent(t_1), parent(t_2), \mathcal{M})$;
while size$(\mathcal{A}) > 0$ **do**
  $(t_1, t_2) \leftarrow remove(\mathcal{A}, 0)$;
  add all pairs of isomorphic nodes of $s(t_1)$ and $s(t_2)$ to $\mathcal{M}$;
  $\mathcal{A} \leftarrow \mathcal{A} \setminus \{(t_1, t_x) \in \mathcal{A}\}$;
  $\mathcal{A} \leftarrow \mathcal{A} \setminus \{(t_x, t_2) \in \mathcal{A}\}$;

VirginiaTech
Invent the Future

AST Mapping with bottom-up and Recovery



Test.java: *destination*

```
public class Test {
   private String foo(int i) {
        if (i == 0) return "Bar";
        else if (i == -1) return "Foo!";
   }
}
```

Test.java: *source*

```
public class Test {
   public String foo(int i) {
      if (i == 0) return "Foo!";
   }
}
```

Tree edit distance Problems

- Between ordered labeled trees
- Minimum-cost sequence of node edit operations that transform one tree into another

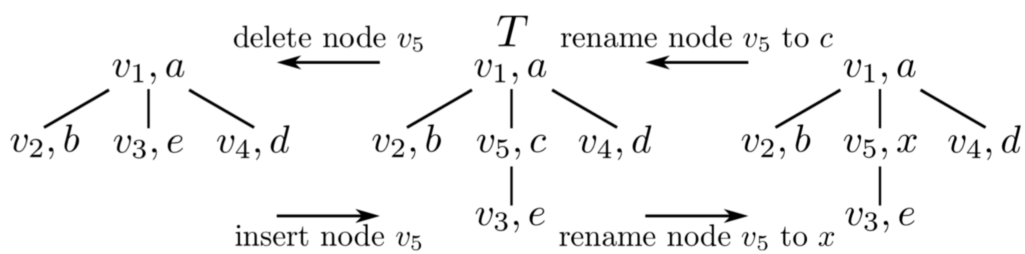
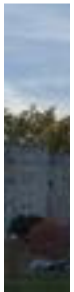
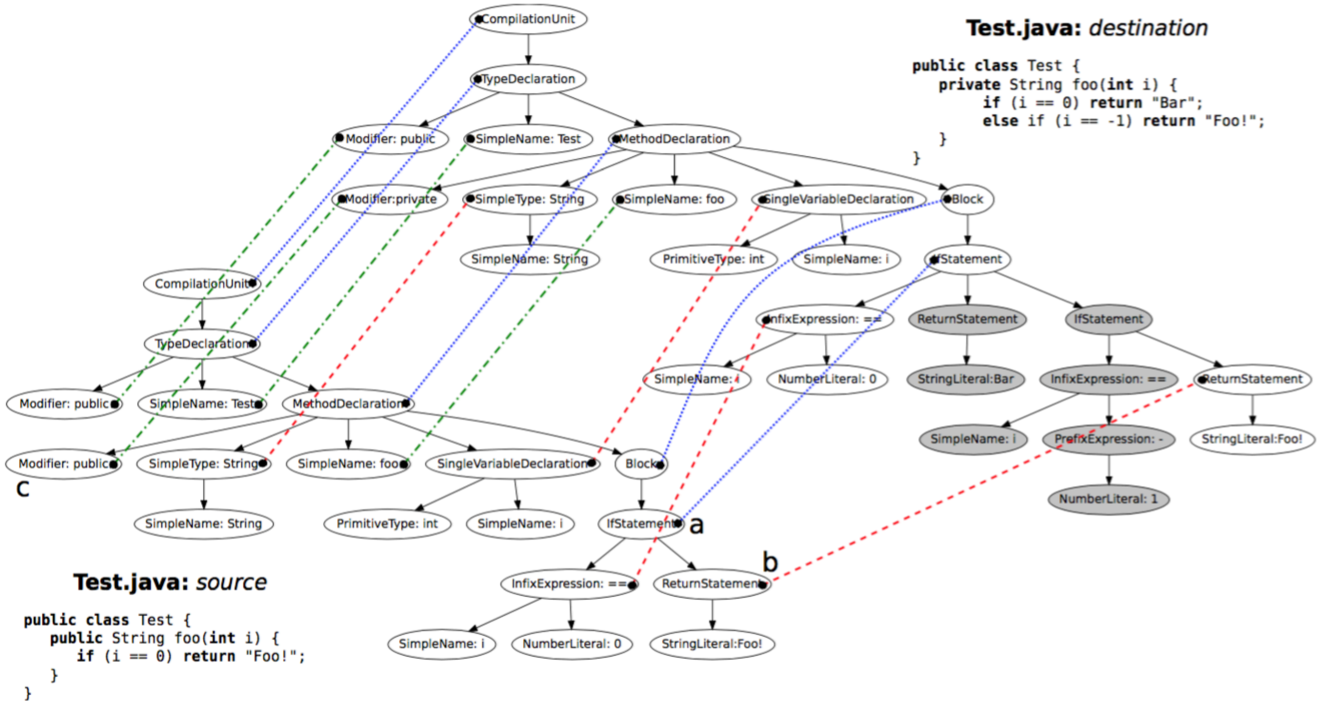


Figure 1: Example trees and edit operations.





Test.java (source)

```
public class Test {
        public String foo(int i) {
                if (i == 0) return "Foo!";
        }
}
```

Test.java (destination)

```
public class Test {
        private String foo(int i) {
                if (i == 0) return "Bar";
                else if (i == -1) return "Foo!";

        }
}
```

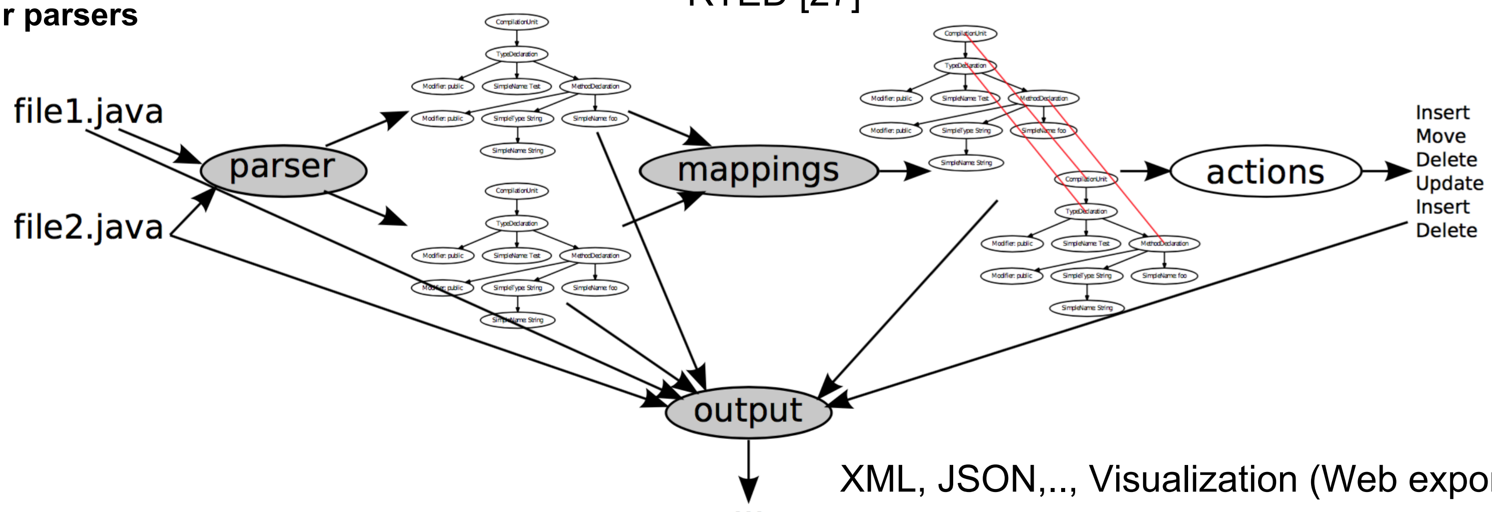
$$add(t_1, a, 1, ReturnStatement, \epsilon)$$
$$add(t_2, t_1, 0, StringLitteral, Bar)$$
$$add(t_3, a, 2, IfStatement, \epsilon)$$
$$add(t_4, t_3, 0, InfixExpression, ==)$$
$$add(t_5, t_4, 0, SimpleName, i)$$
$$add(t_6, t_4, 1, PrefixExpression, -)$$
$$add(t_7, t_6, 0, NumberLiterral, 1)$$
$$move(b, t_3, 1)$$
$$updateValue(c, private)$$

pipe and filter architecture

- Framework(abstract modules in grey)

Java (using the Eclipse JDT parser),
JavaScript (using the Mozilla Rhino parser),
R (using the FastR parser [17]),
C (using the Coccinelle parser [26])
Antlr parsers

GumTree
ChangeDistiller [13]
XYDiff [8]
RTED [27]



XML, JSON,.., Visualization (Web export)

pipe and filter architecture

- Benefit of Antlr support

antlr / grammars-v4

<> Code Issues 62

Grammars written for ANTLR v

1,500 commits

Branch: master New pull re

teverett committed on GitHub

abnf
agc
antlr3
antlr4
apex
arithmetic
asm6502
asn
aspectj
atl
basic
bnf
c
calculator
clf
clif
clojure
cobol85
cpp
creole
csharp
csv
datetime
dcm
dot
ecmascript
erlang
fasta
fol
fortran77
fusion-tables
gff3
gml
golang
graphql
graphstream-dgs
html
icalendar
idl
informix
iri
java
java8
javadoc
jpa
json
kuka
less
logo
lolcode
lua
masm
mdx
memcached_protocol
modelica
modula2pim4
mps
mumath
mumps
muparser
mysql
objc
oncrpc
pascal
pcre
pddl
pdp7
peoplecode
pgn
php
plsql
postalcode
propcalc
protobuf3
python3
r
rcs
redcode
robotwars
ruby
scala
scss
sexpression
sharc
smalltalk
snobol
sparql
sqlite
stacktrace
stringtemplate
suokif
support/bnf2antlr
swift-fin
swift
telephone
tiny
tinyc
tnsnames
tnt
tsql
turtle
ucb-logo
unicode
url
useragent
vb6
vba
verilog
vhdl
wavefront
webidl
xml
xpath

Runtime Performances

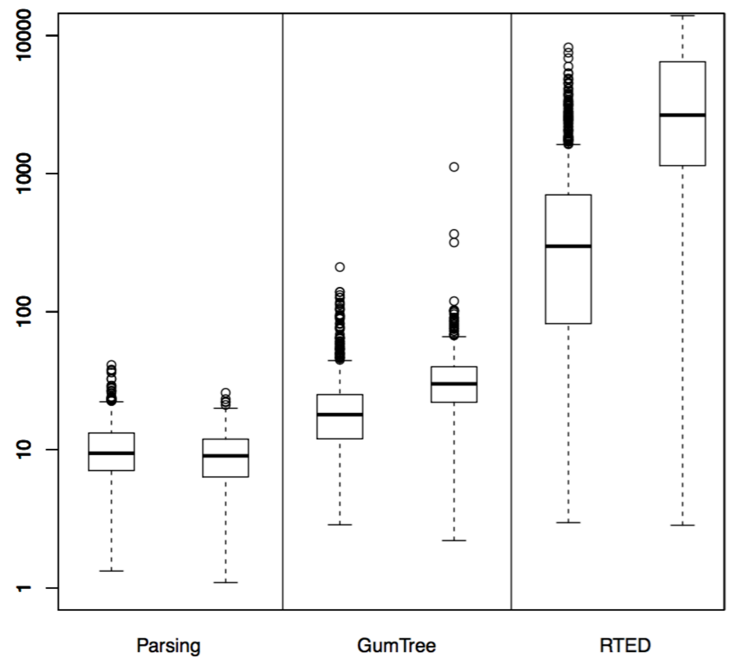
- For Java parser
  - Jenkins 1.509.4 → 1.532.2 where we extracted 1,144 modifications
- For JavaScript parser
  - JQuery 1.8.0 → 1.9.0 where we extracted 650 modifications
- ***Comparisons***
  - ***A classical text diff tool***: *lower bound*, very fast
  - ***GumTree:*** a worst-case complexity of $O(n^2)$
    - the isomorphism test they use is in $O(1)$ thanks to hashcodes
  - ***RTED algorithm****: upper bound,* has a cubic worst-case complexity (n^3)

Discussion Question?

- Why they did NOT compare ***GumTree*** to ***other tree differencing algorithms(**changeDistiller**)*** in running time performance?

Runtime Performances

- Distribution of the running time ratios of the tools



Evaluation

- RQ1)
  - Does ***GumTree*** produce tree differences that are correct and better than ***Unix diff***?
- RQ2)
  - Does ***GumTree*** maximize the number of mappings and minimize the edit script size compared to ***the existing algorithms***?
- RQ3)
  - Does ***GumTree*** detect move operations better than ***ChangeDistiller***?

VirginiaTech Invent the Future

Evaluation(RQ1)

- Manual Evaluation
  - Outputs from *unix diff* and *gumtree* approaches are given to a human evaluator
  - The 144 evaluation items were independently evaluated by **three authors of this paper** called the raters. All 3 raters evaluated all the edit scripts of 144 file pairs at the AST and line level (i.e. 288 outputs). This makes a total of 3 × 2 × 144 = 864 ratings.
  -

- Question #1: *Does GumTree do a good job?*
  The possible answers are:
  1. GumTree does a good job: it helps to understand the change.
  2. GumTree does a bad job.
  3. Neutral.
- Question #2: *Is GumTree better than diff?*
  The possible answers are:
  1. GumTree is better.
  2. diff is better.
  3. GumTree is equivalent to diff.

		Full (3/3)	Majority (2/3)
#1	GT does good job	122	137
	GT does not good job	3	3
	Neutral	0	1
#2	GT better	28	66
	Diff better	3	12
	Equivalent	45	61

Table 1: Agreements of the manual inspection of the 144 transactions by three raters for Question #1 (top) and Question #2 (bottom).

Evaluation(RQ2)

- Measure the performance of tree algorithms with respect to:
  - 1. the number of mappings; 2. the edit script size;
  - ChangeDistiller and RTED
    - ChangeDistiller uses a simplified ASTs where the leaf nodes are code statements. They compute the metrics for both simplified ASTs
    - CDG (ChangeDistiller granularity) and JDTG (Eclipse JDT granularity)

		GT better	CD better	Equiv.
CDG	Mappings	4007 (31.32%)	542 (4.24%)	8243 (64.44%)
	ES size	4938 (38.6%)	412 (3.22%)	7442 (58.18%)
JDTG		GT better	CD better	Equiv.
	Mappings	8378 (65.49%)	203 (1.59%)	4211 (32.92%)
	ES size	10358 (80.97%)	175 (1.37%)	2259 (17.66%)
		GT better	RTED better	Equiv.
	Mappings	2806 (21.94%)	1234 (9.65%)	8752 (68.42%)
	ES size	3020 (23.61%)	2193 (17.14%)	7579 (59.25%)

Table 2: Number of cases where GumTree is better (resp. worse and equivalent) than ChangeDistiller (top, middle) and RTED (bottom) for 2 metrics, number of mappings and edit script size (ES size), at the CDG granularity (top) and JDTG granularity (middle, bottom).

VirginiaTech Invent the Future

Evaluation(RQ3)

- Analysis of Move Actions
  - GumTree and ChangeDistiller

18 instances more in CD
49 instances more in GT

	GT only move op	GT other op
CD only move op	77	1
CD other op	52	12662

<-CD produces only move actions, and GT other actions

GT produces only move actions, And CD other actions

Table 3: Comparison of the number of move operations from GumTree and ChangeDistiller for 12 792 file pairs to be compared.

Discussion Question

- Some possible weakness or remaining experiment in RQ1-2-3?

VirginiaTech
Invent the Future

Discussion Question

- Analysis II: Discuss about ideas or thoughts the paper provoked?
  - the new problems you identify?
  - new research directions inspired?

Thank you!!

VirginiaTech
Invent the Future