Proof Re-Use in Java Software Verification

Vladimir Klebanov
Universität Koblenz

June 8, 2004

Goal: Proof Re-Use for Java Software Verification

Why Re-Use Proofs?

Typical use case: verification **fails**.
➥ the program (the specification?) has to be amended; user starts verification from scratch...

A re-use facility would recycle unaffected proof parts, saving efforts, user interaction.

The Re-Use Task

Bird’s eye view:

Frontend: Source	Backend: Proof
old source	old proof
new source	?

Some Re-Use Scenarios

What can happen?

- statement insertion
- statement deletion
- change in expression
- ...

Imagine

You want to “add a case”...

Related Efforts

Different approaches to re-use

- Abstraction: proof planning, analogy reasoners
- Construction: KIV
- Incremental fixed: Isabelle
- Incremental similarity-based: us

What we don't want to do

- Learn from proofs in general

The Re-Use Framework

Keep a list of **candidates** $\hat{=}$ marked nodes in template

❶ Match candidates against open goals ➥ possible **re-use units**

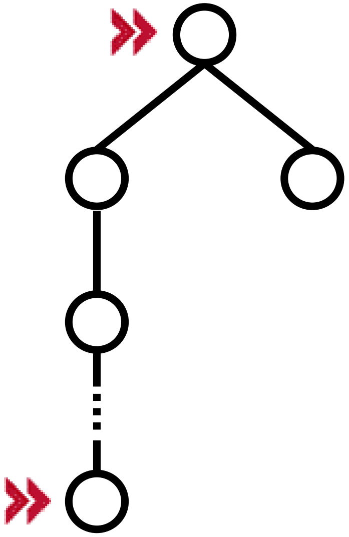
❷ Select re-use unit with best score, apply it

❸ Advance markers in the template proof

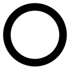
Questions:

Where do the candidates originally come from?
What if nothing works?
Where does the new proof "stuff" come from?

In Action

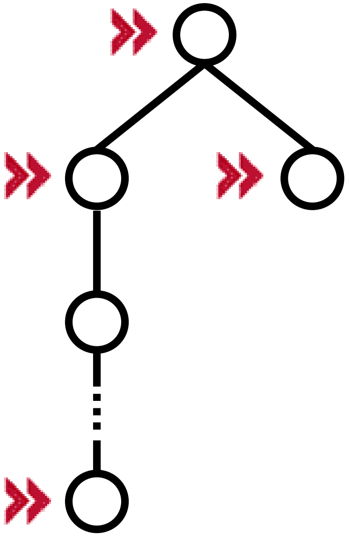


Template

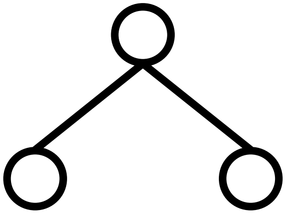


Target

In Action

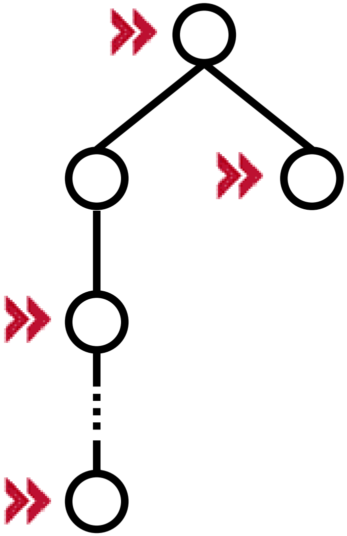


Template

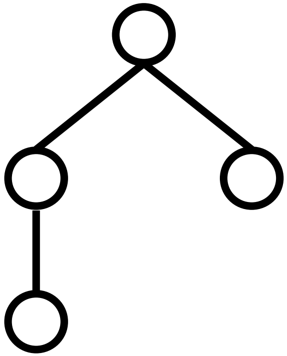


Target

In Action



Template



Target

Formula Similarity Function

Beyond copy & paste: Identify similar situations by comparing rule application foci.

Three cases:

- Symbolic execution rules: Eugene Myers **diff** algorithm on the programs in focus
- First-order rules
- Focus-less rules

We install a **cut-off threshold**.

Program Abstractions

From programs

$$\alpha : \left\{ \begin{array}{l} \texttt{int x; int result;} \\ \texttt{result = x/x;} \end{array} \right.$$

$$\beta : \left\{ \begin{array}{l} \texttt{int x; int result;} \\ \texttt{if (x==0) result=1; else result=x/x;} \end{array} \right.$$

To sequences of statement signatures

$$A : \left\{ \begin{array}{l} \texttt{LocalVarDecl}, \texttt{LocalVarDecl}, \\ \texttt{Assignment(int)} \end{array} \right.$$

$$B : \left\{ \begin{array}{l} \texttt{LocalVarDecl}, \texttt{LocalVarDecl}, \\ \underline{\texttt{If}}, \texttt{Assignment(int)}, \underline{\texttt{Assignment(int)}} \end{array} \right.$$

Program Similarity Function

Let $E(A, B) = e_1\, e_2 \cdots e_n$ be the minimal edit script for the abstractions A, B.

The *similarity score* of α, β:

$$\delta(\alpha, \beta) = \delta(A, B) \quad = \quad - \sum_{e_i \in E(A,B)} P(e_i)$$

where the penalty $P(e)$ for an edit command e is

$$P(e) \quad = \quad \begin{cases} \displaystyle\sum_{k=1}^{t} \frac{0.75}{x+k} & \text{if } e = x\; I\; b_1\, b_2 \cdots b_t \\ \displaystyle\frac{1}{x+1} & \text{if } e = x\; D \end{cases}$$

First-Order Formula Similarity

❶ abstraction step

❷ compare foci

❸ difference detection on whole formulas

❹ compare focus position in formula

Augmenting With Connectivity

Introduce parent relationship for formulae. Prefer proof steps that respect it.

Feedback loop: amplifies good decisions... unfortunately bad decisions too.

Prevents related proof steps being torn apart.

On the Frontend

Obtain source code diffs from CVS.

Mark statement after each difference hunk.

```
  int x;
  int result;
+ if(x==0) {
+    result=1;
+ } else {
     result=x/x;
+ }
```

Why This Works

❶ Proof structure follows program structure

❷ Similar situations warrant similar actions

❸ Calculus to a high degree "locally deterministic"

- ② Symbolic execution rules only applicable to the active statement
- ③ No split rule ➥ active statements do not multiply

At Last

DEMO

TOC