Applications of Metatheory: Verification of Compiler Optimisations

Richard Warburton & Sara Kalvala
University of Warwick, Formal Methods Group

October 17, 2007

Objectives and Motivation

- Modern Compilers Perform Complex Optimisations

Objectives and Motivation

- Modern Compilers Perform Complex Optimisations
- Bugs within a compiler potentially viral

Objectives and Motivation

- Modern Compilers Perform Complex Optimisations
- Bugs within a compiler potentially viral
- Idea: Formal Methods useful, Problem: how to approach?

Our Approach

- Domain Specific Language called TRANS (Lacey)

Our Approach

- Domain Specific Language called TRANS (Lacey)
- Formal Semantics of TRANS

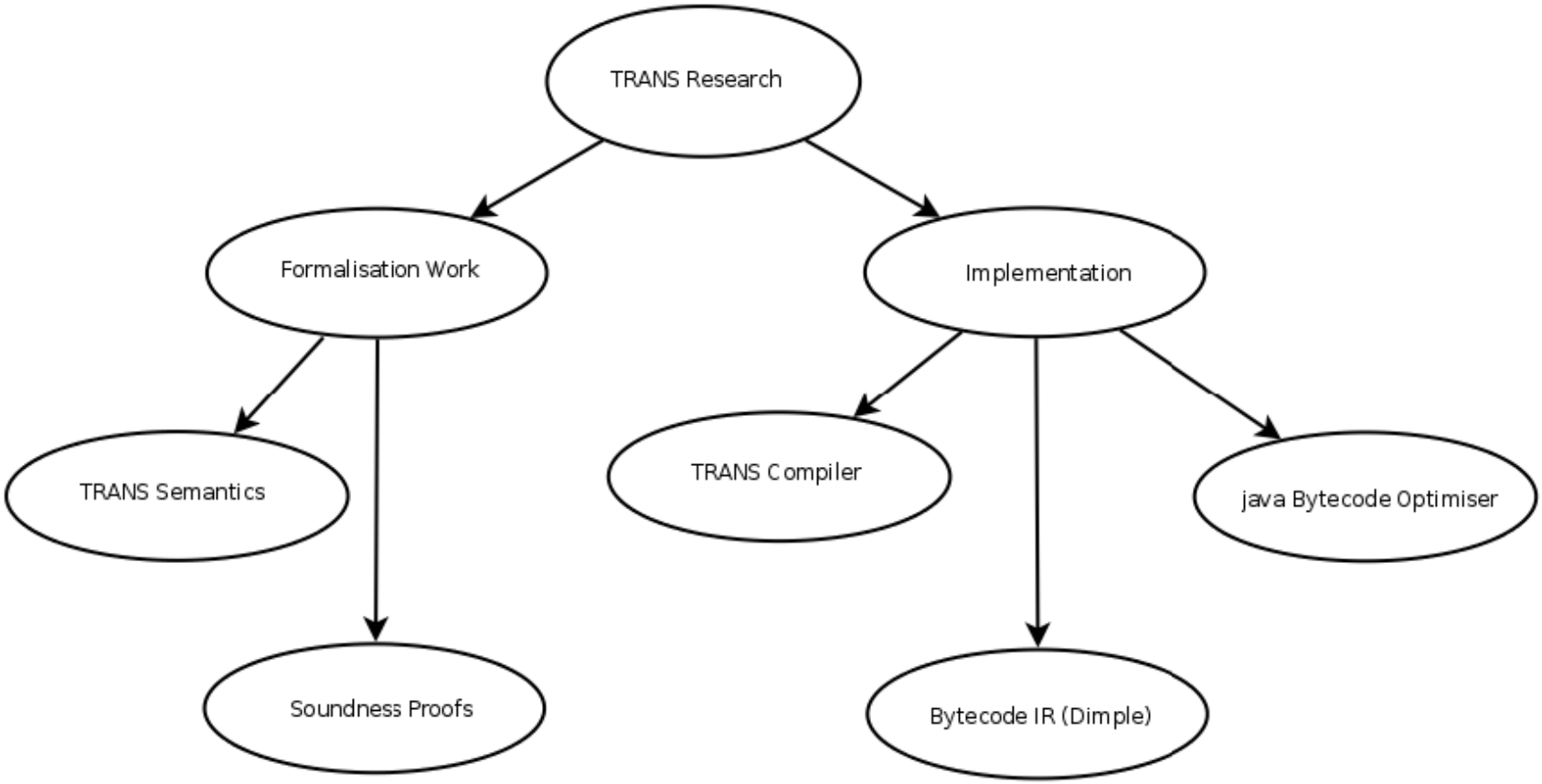
Our Approach

- Domain Specific Language called TRANS (Lacey)
- Formal Semantics of TRANS
- Formal Semantics of language to be optimised

Our Approach

- Domain Specific Language called TRANS (Lacey)
- Formal Semantics of TRANS
- Formal Semantics of language to be optimised
- Prove Soundness of transformations

Architecture - Overview



Example - Loop Invariant Code Motion

Idea: move operations that are invariant of the loop iteration out of it

Example - Loop Invariant Code Motion

Idea: move operations that are invariant of the loop iteration out of it

Before:

```
for(int i = 0;i<10;i++) {
        y = z*5;
        x += y*i;
}
```

Example - Loop Invariant Code Motion

Idea: move operations that are invariant of the loop iteration out of it

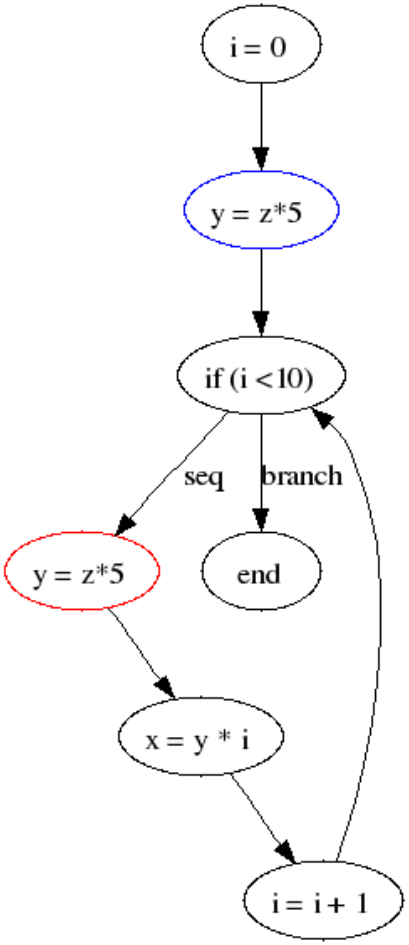
Before:

```
for(int i = 0;i<10;i++) {
        y = z*5;
        x += y*i;
}
```

After:

```
y = z*5;
for(int i = 0;i<10;i++) {
        x += y*i;
}
```

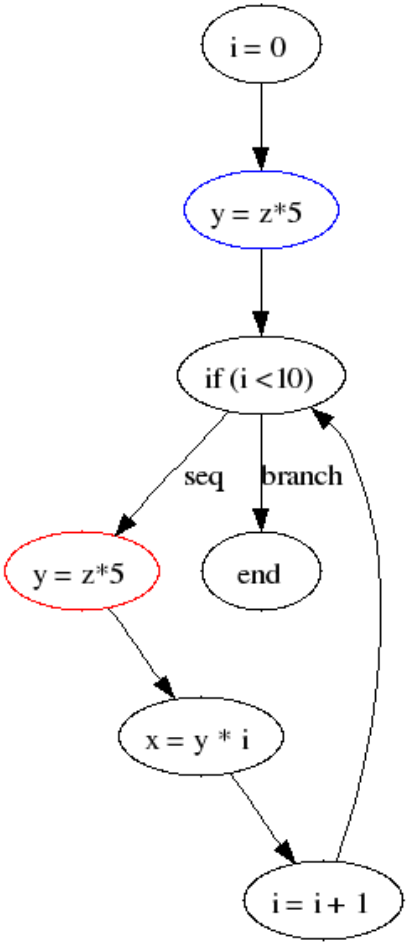
TRANS - Loop Invariant Hoisting



after : skip $\Rightarrow$ x := e

before : x := e $\Rightarrow$ skip

TRANS - Loop Invariant Hoisting



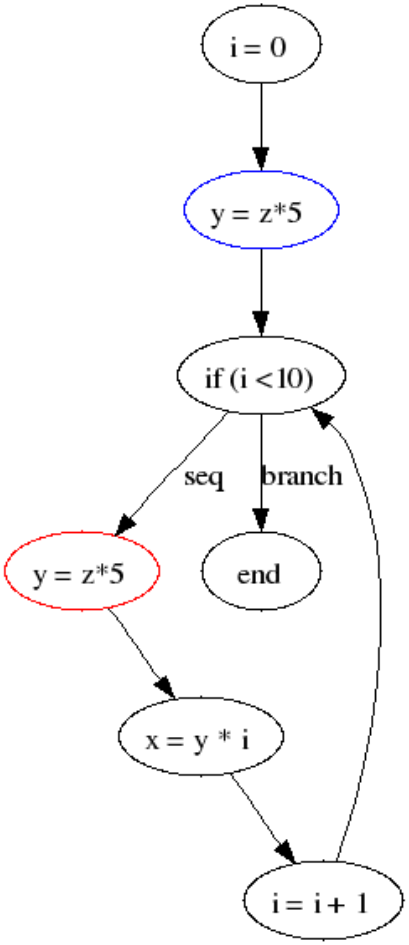
after : skip $\Rightarrow$ x := e

before : x := e $\Rightarrow$ skip

if

A ($\neg$ use(x) W node(before)) @ after

TRANS - Loop Invariant Hoisting



after : skip $\Rightarrow$ x := e

before : x := e $\Rightarrow$ skip

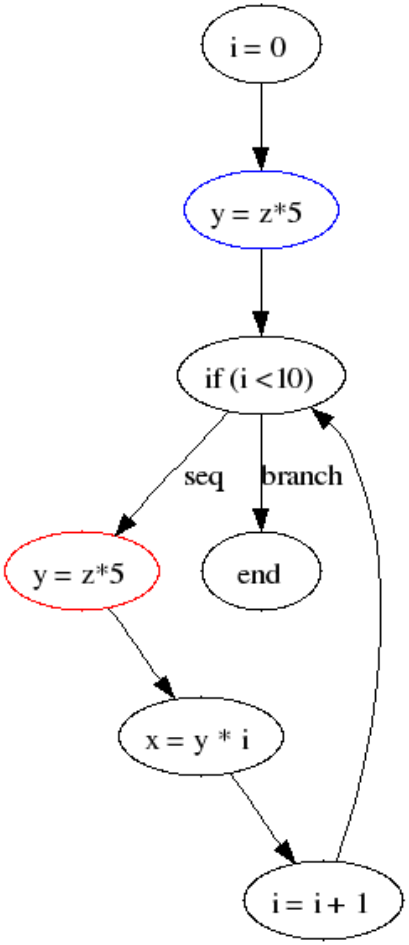
if

$\mathtt{A}\ (\neg\ \mathtt{use(x)}\ \mathtt{W}\ \mathtt{node(before)})\ @\ \mathrm{after}$

$(\neg\ \mathtt{use(x)} \wedge \overleftarrow{\mathtt{A}}((\neg\ \mathtt{def(x)} \vee \mathtt{node(before)}\ ) \wedge$

$\mathtt{trans(e)}\ \mathtt{W}\ \mathtt{node(after)})\ @\ \mathrm{before}$

TRANS - Loop Invariant Hoisting



after : skip $\Rightarrow$ x := e

before : x := e $\Rightarrow$ skip

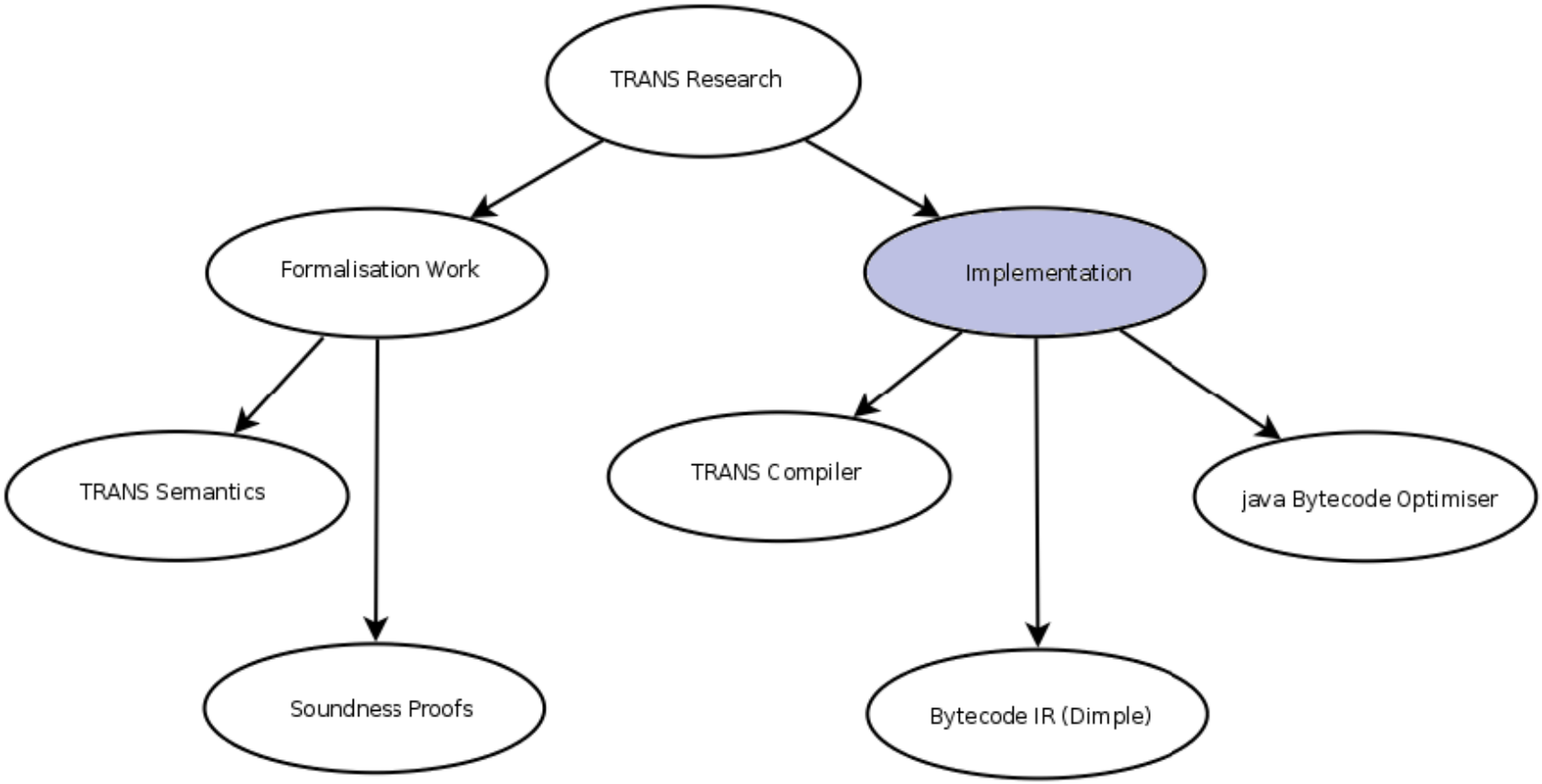
if

$\mathtt{A}\ (\neg\ \mathtt{use(x)}\ \mathtt{W}\ \mathtt{node(before)})\ @\ \mathrm{after}$

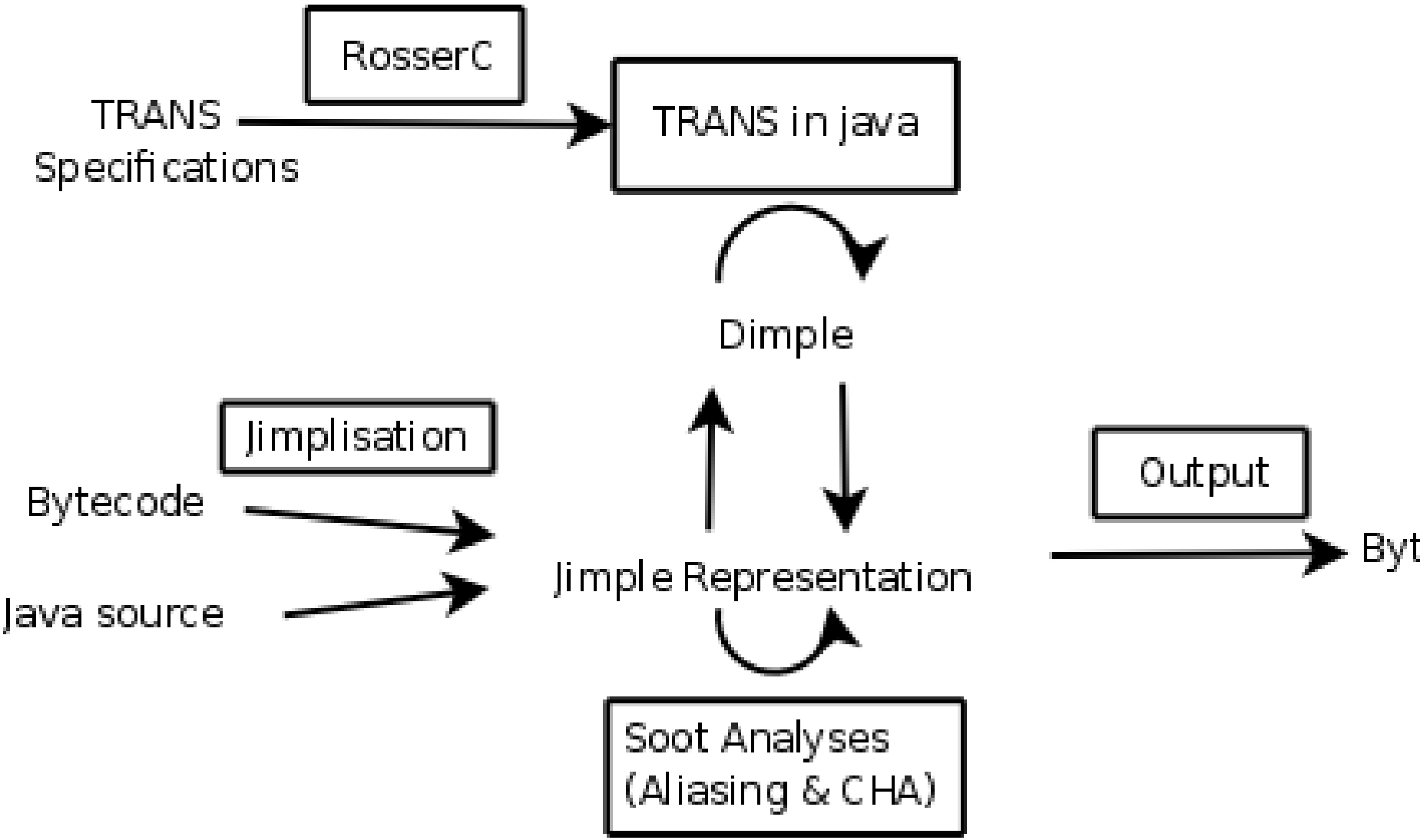
$(\neg\ \mathtt{use(x)} \wedge \overleftarrow{\mathtt{A}}((\neg\ \mathtt{def(x)} \vee \mathtt{node(before)}\ ) \wedge$

$\mathtt{trans(e)}\ \mathtt{W}\ \mathtt{node(after)})\ @\ \mathrm{before}$

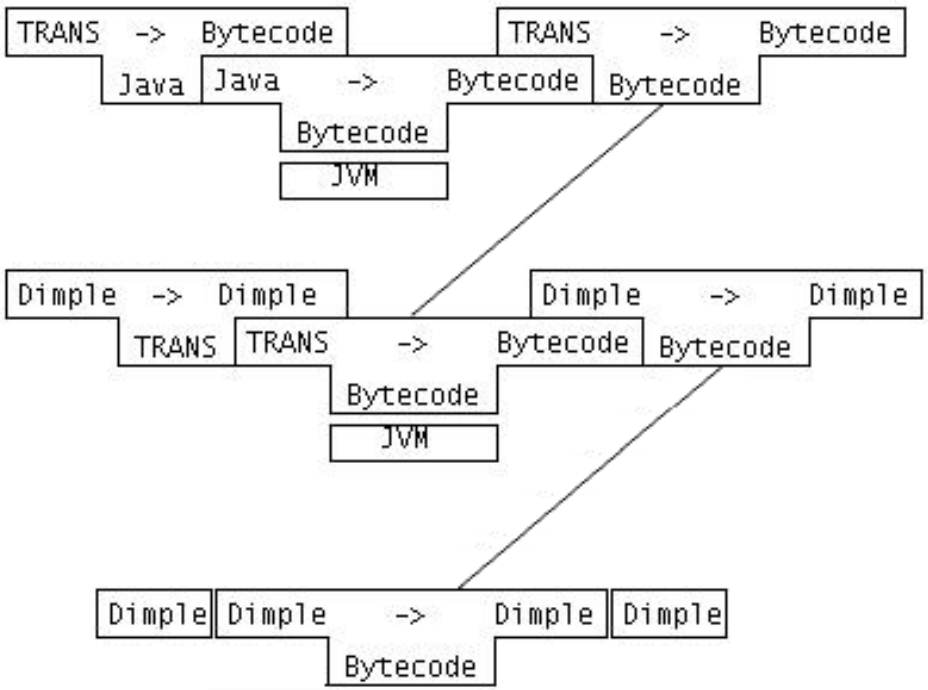
Architecture - Overview



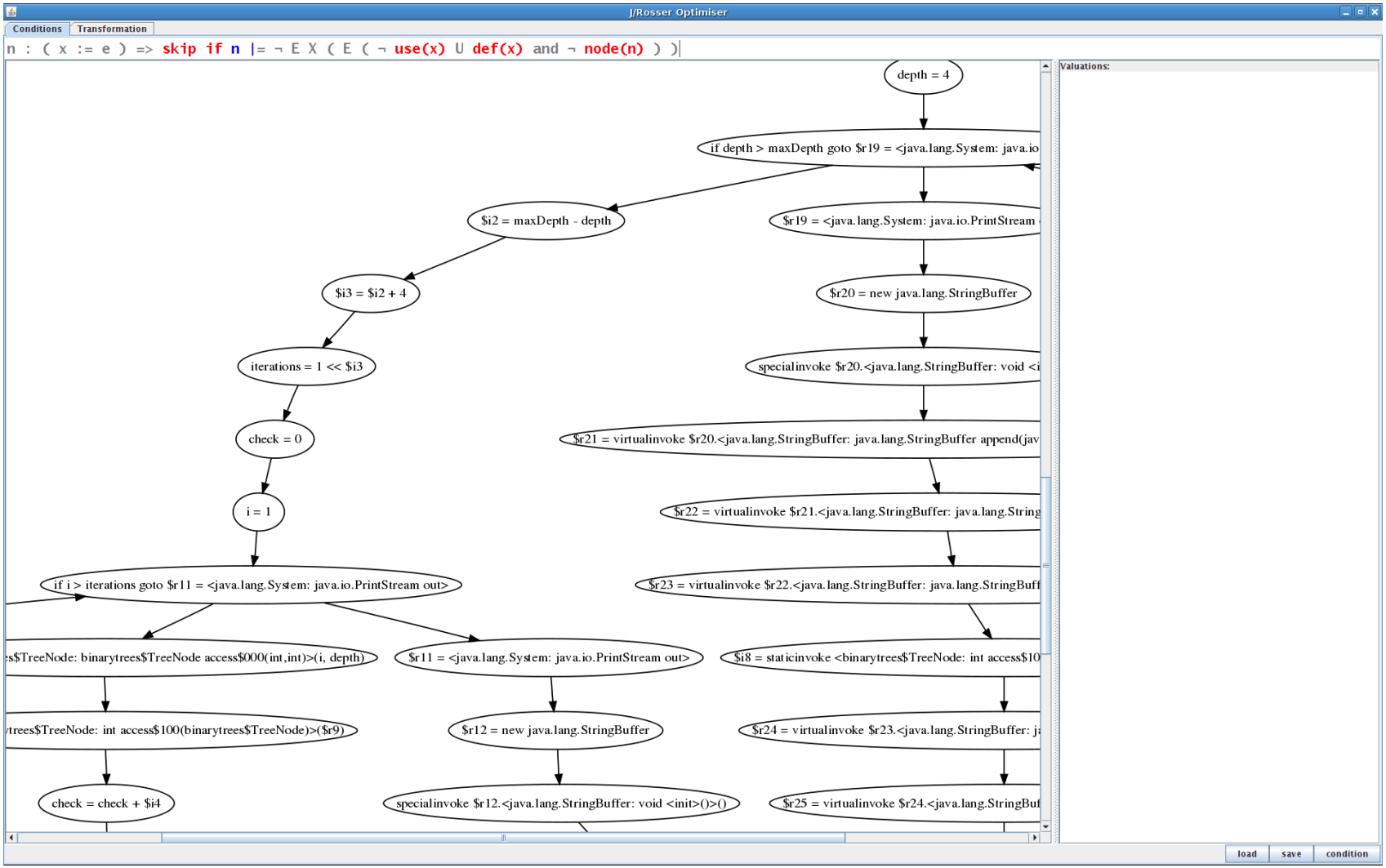
Architecture



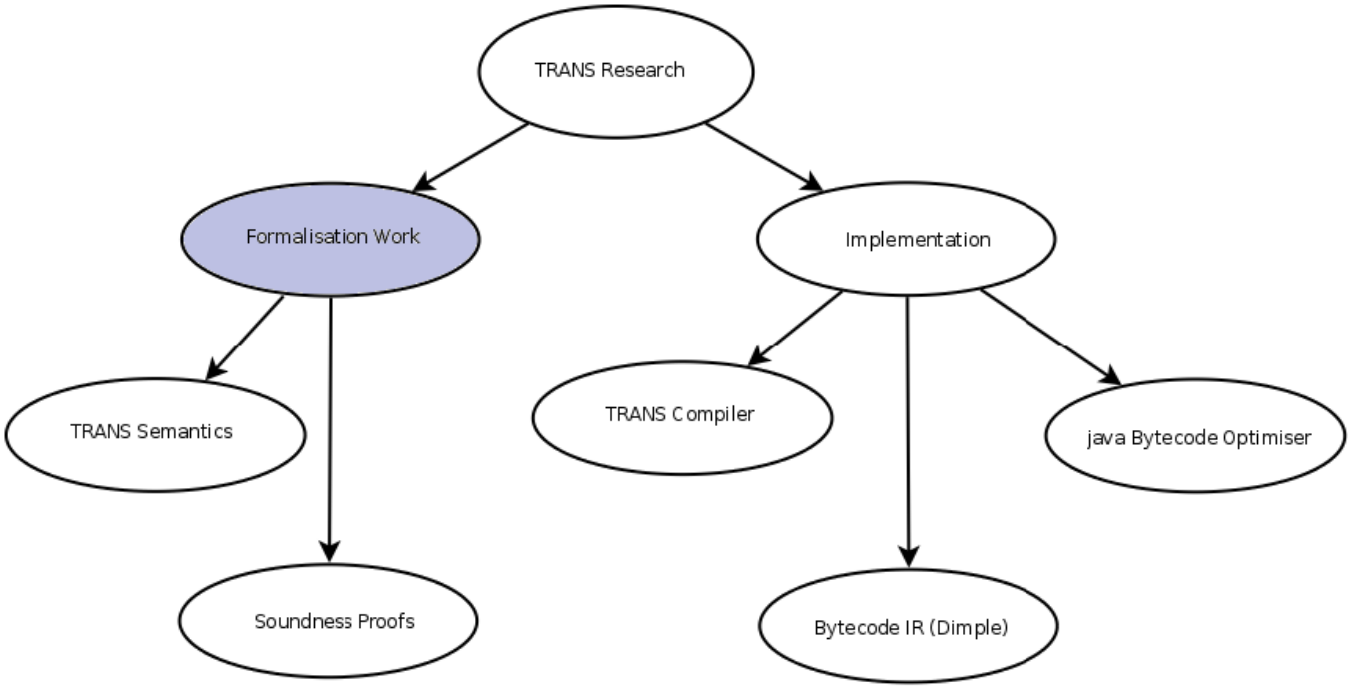
T-Diagram


TRANS -> Bytecode
Java
Java -> Bytecode
Bytecode
JVM
TRANS -> Bytecode
Bytecode
Dimple -> Dimple
TRANS
TRANS -> Bytecode
Bytecode
JVM
Dimple -> Dimple
Bytecode
Dimple
Dimple -> Dimple
Dimple
Bytecode

Current Program



Formalisation - Overview



Formalisation

Layers:

- Isabelle/HOL (Paulson, Nipkow et al.)

Formalisation

Layers:

- Isabelle/HOL (Paulson, Nipkow et al.)
- Jinja (Nipkow, Klein)

Formalisation

Layers:

- Isabelle/HOL (Paulson, Nipkow et al.)
- Jinja (Nipkow, Klein)
- Control Flow Graph

Formalisation

Layers:

- Isabelle/HOL (Paulson, Nipkow et al.)
- Jinja (Nipkow, Klein)
- Control Flow Graph
- TRANS Semantics

Proof Approach

- Work in progress

Proof Approach

- Work in progress
- Soundness of an Optimisation is semantic equivalence between initial and transformed programs.

Proof Approach

- Work in progress
- Soundness of an Optimisation is semantic equivalence between initial and transformed programs.
- Source and Transformed Programs members of a bisimulation relation.

Proof Approach

- Work in progress
- Soundness of an Optimisation is semantic equivalence between initial and transformed programs.
- Source and Transformed Programs members of a bisimulation relation.
- but Bisimulation between Kripke structures? Idea: use CFG.

Proof Approach

- Work in progress
- Soundness of an Optimisation is semantic equivalence between initial and transformed programs.
- Source and Transformed Programs members of a bisimulation relation.
- but Bisimulation between Kripke structures? Idea: use CFG.
- Intra-procedural Optimisations, so CFG of a given method.

Proof Approach

- Work in progress
- Soundness of an Optimisation is semantic equivalence between initial and transformed programs.
- Source and Transformed Programs members of a bisimulation relation.
- but Bisimulation between Kripke structures? Idea: use CFG.
- Intra-procedural Optimisations, so CFG of a given method.
- Bisimulation within Isabelle/HOL - Co-induction.

Related Work

- initial TRANS (Lacey)

Related Work

- initial TRANS (Lacey)
- Cobalt, Rhodium (Lerner et al.)

Related Work

- initial TRANS (Lacey)
- Cobalt, Rhodium (Lerner et al.)
- TTL (Kanade et al)

Related Work

- initial TRANS (Lacey)
- Cobalt, Rhodium (Lerner et al.)
- TTL (Kanade et al)
- TV, Credible Compilation

Ongoing work

- Complete Implementation

Ongoing work

- Complete Implementation
- Finish Equivalence Proof Tactics

Ongoing work

- Complete Implementation
- Finish Equivalence Proof Tactics
- Inter-procedural optimisation

Conclusions

- Implementation easier when informed by theory.

Conclusions

- Implementation easier when informed by theory.
- nature of language metatheory definitions influential when built upon.

Conclusions

- Implementation easier when informed by theory.
- nature of language metatheory definitions influential when built upon.
- eg: within Jinja single step execution and rtc. help definition of CFG

The End

Questions?