Applications of Metatheory: Verification of Compiler Optimisations

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

October 17, 2007
Modern Compilers Perform Complex Optimisations
Objectives and Motivation

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

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

Richard Warburton, University of Warwick
Applications of Metatheory: Verification of Compiler Optimisations
Example - Loop Invariant Code Motion

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

Before:

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

After:

```c
y = z * 5;
for (int i = 0; i < 10; i++) {
    x += y * i;
}
```
TRANS - Loop Invariant Hoisting

\[
\begin{align*}
\text{after} & : \text{skip} \Rightarrow x := e \\
\text{before} & : x := e \Rightarrow \text{skip}
\end{align*}
\]
TRANS - Loop Invariant Hoisting

after: skip \Rightarrow x := e
before: x := e \Rightarrow skip
if
A \ (\neg \text{use}(x) \ W \ \text{node(before))) \ @ \ after
TRANS - Loop Invariant Hoisting

after : skip ⇒ x := e
before : x := e ⇒ skip
if
A (¬ use(x) W node(before)) @ after
(¬ use(x) ∧ A ((¬ def(x) ∨ node(before)) ) ∧
trans(e) W node(after)) @ before
TRANS - Loop Invariant Hoisting

after : skip \Rightarrow x := e
before : x := e \Rightarrow skip

if
A \leftarrow use(x) W node(before)) @ after
\neg use(x) \land \neg def(x) V node(before) \land trans(e) W node(after)) @ before
Introduction

Architecture - Overview

TRANS Research

- Formalisation Work
  - TRANS Semantics
  - Soundness Proofs

- Implementation
  - TRANS Compiler
  - java Bytecode Optimiser
  - Bytecode IR (Dimple)
Introduction

T-Diagram

Richard Warburton, University of Warwick

Applications of Metatheory: Verification of Compiler Optimisations
Introduction

Current Program

Richard Warburton, University of Warwick
Applications of Metatheory: Verification of Compiler Optimisations
Formalisation - Overview

- TRANS Research
  - Formalisation Work
    - TRANS Semantics
      - Soundness Proofs
    - TRANS Compiler
      - Bytecode IR (Dimple)
  - Implementation
    - java Bytecode Optimiser
Layers:

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

Layers:
- Isabelle/HOL (Paulson, Nipkow et al.)
- Jinja (Nipkow, Klein)
- Control Flow Graph
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.
Work in progress

Soundness of an Optimisation is semantic equivalence between initial and transformed programs.

Source and Transformed Programs members of a bisimulation relation.
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.
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.
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
Implementation easier when informed by theory.
Implementation easier when informed by theory.

nature of language metatheory definitions influential when built upon.
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
Questions?