Optimisation validation

Lennart Beringer

Lehrstuhl für Theoretische Informatik
Ludwig-Maximilians-Universität München
Joint work with Alberto Momigliano and David Aspinall,
LFCS, University of Edinburgh

COCV, April 2nd, 2006

Funded by EPSRC project ReQueST (EP/C537068/1) and EU-project
Mobius (IST-2005-015905)
Motivation

Long-term goal

Compiler (modules) that produce evidence that emitted code satisfies certain properties.

Properties:

- Syntactic well-formedness
- Semantic well-formedness: type-correctness (e.g. bytecode verification)
- Satisfaction of memory access policies (PCC, TAL)
- Functional correctness w.r.t. input program (verified compiler, translation validation)
- Resource behaviour (cf. invited talk):
  - absolute (satisfaction of bounds on time, space,...)
  - relative (improvement w.r.t. input program)
Basic requirement

Verification infrastructure that allows us to express and prove resource specifications w.r.t. a variety of cost models.

Our contribution

Base-level infrastructure, comprising

- a general purpose program logic (expressive, little automation) for functional and non-functional properties, with
- a flexible notion of structured cost models ("resource algebras")
- validation of concrete example transformations w.r.t. resource behaviour
Basic requirement

Verification infrastructure that allows us to express and prove resource specifications w.r.t. a variety of cost models.

Our contribution

Base-level infrastructure, comprising

- a general purpose program logic (expressive, little automation) for functional and non-functional properties, with
- a flexible notion of structured cost models ("resource algebras")
- validation of concrete example transformations w.r.t. resource behaviour
Basic requirement

Verification infrastructure that allows us to express and prove resource specifications w.r.t. a variety of cost models.

Our contribution

Base-level infrastructure, comprising

- a general purpose program logic (expressive, little automation) for functional and non-functional properties, with
  - a flexible notion of structured cost models ("resource algebras")
  - validation of concrete example transformations w.r.t. resource behaviour
Basic requirement

Verification infrastructure that allows us to express and prove resource specifications w.r.t. a variety of cost models.

Our contribution

Base-level infrastructure, comprising

- a general purpose program logic (expressive, little automation) for functional and non-functional properties, with
- a flexible notion of structured cost models ("resource algebras")
- validation of concrete example transformations w.r.t. resource behaviour
Basic requirement

Verification infrastructure that allows us to express and prove resource specifications w.r.t. a variety of cost models.

Our contribution

Base-level infrastructure, comprising

- a general purpose program logic (expressive, little automation) for functional and non-functional properties, with
- a flexible notion of structured cost models ("resource algebras")
- validation of concrete example transformations w.r.t. resource behaviour
Language & operational semantics

- “Beautified” (well-structured) JVM: Grail
- ANF-style let bindings (simple expressions may alter the heap), function calls in tail position, general recursion via method invocation
- Reversible expansion into JVM
- Big-step evaluation relation $E \vdash h, e \Downarrow h', v, r$

Example rule:

$$
E \vdash h, e_1 \Downarrow h_1, v_1, r_1 \quad E\langle x := v_1 \rangle \vdash h_1, e_2 \Downarrow h_2, v, r_2
$$

\[ E \vdash h, \text{let } x = e_1 \text{ in } e_2 \Downarrow h_2, v, r_1 + R^{\text{let}} + r_2 \]
Resource algebras (dynamic)

Structural (i.e. syntax-oriented) cost models:

- Carrier set $R$, with order $\leq_R$
- Constants for all syntax-formers of Grail,
- Application of operators fixed in operational semantics ("non-invasiveness") $\sim$ uniform cost-accounting
- Instantiation results in program logics for desired cost model:
  - Examples: counters (instructions, allocations, ...), maximal frame depth, flags (parameter values), traces

Alternative
Program instrumentation: invasive?, traces of semantic objects?
A resource algebra $\mathcal{R}$ is a partially ordered monoid $(R, 0, +, \leq)$, i.e. $(R, 0, +)$ is a monoid and $(R, \leq)$ a partially ordered set, where

1. 0 is the minimum element,
2. + is order preserving on both sides.

Moreover, $\mathcal{R}$ has constants in $R$ for each expression former: $\mathcal{R}^{\text{int}}, \mathcal{R}^{\text{null}}, \mathcal{R}^{\text{var}}, \mathcal{R}^{\text{prim}}, \mathcal{R}^{\text{new}}, \mathcal{R}^{\text{getf}}, \mathcal{R}^{\text{putf}}, \mathcal{R}^{\text{comp}}, \mathcal{R}^{\text{let}}, \mathcal{R}^{\text{if}}, \mathcal{R}^{\text{call}}$ and a monotone operator $\mathcal{R}^{\text{meth}}_{C,m,v} : R \rightarrow R$. 
<table>
<thead>
<tr>
<th>$\mathcal{R}$</th>
<th>Time</th>
<th>Frames</th>
<th>MethCnsts</th>
<th>MethFreq$^l$</th>
<th>MethGuard</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>\mathcal{R}</td>
<td>$</td>
<td>$N$</td>
<td>$N$</td>
<td>$\mathcal{M}S(Id)$</td>
</tr>
<tr>
<td>$\mathcal{R}_{int}^i$</td>
<td>1</td>
<td>0</td>
<td>$\emptyset$</td>
<td>(1, 0)</td>
<td>tt</td>
</tr>
<tr>
<td>$\mathcal{R}_{null}$</td>
<td>1</td>
<td>0</td>
<td>$\emptyset$</td>
<td>(1, 0)</td>
<td>tt</td>
</tr>
<tr>
<td>$\mathcal{R}_{var}$</td>
<td>1</td>
<td>0</td>
<td>$\emptyset$</td>
<td>(1, 0)</td>
<td>tt</td>
</tr>
<tr>
<td>$\mathcal{R}_{prim}$</td>
<td>1</td>
<td>0</td>
<td>$\emptyset$</td>
<td>(1, 0)</td>
<td>tt</td>
</tr>
<tr>
<td>$\mathcal{R}_{new}$</td>
<td>3</td>
<td>0</td>
<td>$\emptyset$</td>
<td>(3, 0)</td>
<td>tt</td>
</tr>
<tr>
<td>$\mathcal{R}_{getf}$</td>
<td>2</td>
<td>0</td>
<td>$\emptyset$</td>
<td>(2, 0)</td>
<td>tt</td>
</tr>
<tr>
<td>$\mathcal{R}_{putf}$</td>
<td>3</td>
<td>0</td>
<td>$\emptyset$</td>
<td>(3, 0)</td>
<td>tt</td>
</tr>
<tr>
<td>$\mathcal{R}_{comp}$</td>
<td>0</td>
<td>0</td>
<td>$\emptyset$</td>
<td>(0, 0)</td>
<td>tt</td>
</tr>
<tr>
<td>$\mathcal{R}_{let}$</td>
<td>1</td>
<td>0</td>
<td>$\emptyset$</td>
<td>(1, 0)</td>
<td>tt</td>
</tr>
<tr>
<td>$\mathcal{R}_{if}$</td>
<td>0</td>
<td>0</td>
<td>$\emptyset$</td>
<td>(0, 0)</td>
<td>tt</td>
</tr>
<tr>
<td>$\mathcal{R}_{call}$</td>
<td>1</td>
<td>0</td>
<td>$\emptyset$</td>
<td>(1, 0)</td>
<td>tt</td>
</tr>
<tr>
<td>$\mathcal{R}<em>{meth}^C</em>{m,v}(r)$</td>
<td>$</td>
<td>v</td>
<td>+ 2 + r$</td>
<td>$r + 1$</td>
<td>$r \cup {C.m}$</td>
</tr>
<tr>
<td>$0_\mathcal{R}$</td>
<td>0</td>
<td>0</td>
<td>$\emptyset$</td>
<td>(0, 0)</td>
<td>tt</td>
</tr>
<tr>
<td>$+_\mathcal{R}$</td>
<td>$max$</td>
<td>$\cup_+$</td>
<td>$+_\text{Freq}$</td>
<td>$\land$</td>
<td></td>
</tr>
<tr>
<td>$\leq_\mathcal{R}$</td>
<td>$\leq$</td>
<td>$\subseteq_+$</td>
<td>$\leq_\text{Freq}$</td>
<td>$\leq_\text{Guard}$</td>
<td></td>
</tr>
</tbody>
</table>
Motivation: single verification of loop bodies & method bodies

**Form of judgements:** $G \triangleright e : P$

- $P$ is an assertion (predicate in meta-logic) over all semantic components $(E, h, h', v, r)$
- $G$ is proof context, used for verification of recursive phrases

Rules formulated a la VDM, without pre-conditions:

$$
\begin{align*}
\Gamma \triangleright e_1 : A & \quad \Gamma \triangleright e_2 : B \\
\Gamma \triangleright \text{let } x = e_1 \text{ in } e_2 & \\
: \quad \lambda E h h' v r. \exists r_1 r_2 h_1 w. \lambda A E h h' v r. \lambda \neg \perp \wedge B (E(x := w)) h_1 h' v r_2 \wedge \\
& \quad r = r_1 + R_{\text{let}} + r_2
\end{align*}
$$

(VLET)
Meta-theoretic properties

Interpretation

Partial correctness:

\[ \models e : P \iff \forall E \ h \ h' \ v \ r. \ E \vdash h, e \downarrow h', v, r \rightarrow P(E, h, h', v, r) \]

Theorem

Soundness: \( \emptyset \triangleright e : P \) implies \( \models e : P \)

Theorem

(Relative) completeness: \( \models e : P \) implies \( \emptyset \triangleright e : P \)

Proofs formalised in Isabelle/HOL.
Meta-theoretic properties

Interpretation

Partial correctness:

\[
\models e : P \iff \\
\forall E \ h \ h' \ v \ r. E \vdash h, e \downarrow h', v, r \rightarrow P(E, h, h', v, r)
\]

Theorem

Soundness: \( \emptyset \triangleright e : P \) implies \( \models e : P \)

Theorem

(Relative) completeness: \( \models e : P \) implies \( \emptyset \triangleright e : P \)

Proofs formalised in Isabelle/HOL.

Lennart Beringer

Optimisation validation
Meta-theoretic properties

Interpretation

Partial correctness:

\[\models e : P \iff \forall E \ h \ h' \ v \ r. \ E \vdash h, e \downarrow h', v, r \rightarrow P(E, h, h', v, r)\]

Theorem

Soundness: \(\emptyset \triangleright e : P\) implies \(\models e : P\)

Theorem

(Relative) completeness: \(\models e : P\) implies \(\emptyset \triangleright e : P\)

Proofs formalised in Isabelle/HOL.
Meta-theoretic properties

**Interpretation**

Partial correctness:

\[ \models e : P \iff \forall E \ h \ h' \ v \ r. \ E \vdash h, e \downarrow h', v, r \rightarrow P(E, h, h', v, r) \]

**Theorem**

Soundness: \( \emptyset \triangleright e : P \) implies \( \models e : P \)

**Theorem**

(Relative) completeness: \( \models e : P \) implies \( \emptyset \triangleright e : P \)

Proofs formalised in Isabelle/HOL.
Optimisations

- Single optimisation step (operational):
  \[ E \vdash h, e_1 \downarrow h_1, \nu, r \land E \vdash h, e_2 \downarrow h'_1, w, q \implies q \leq_R r \]
- Chain of optimisation steps \( e_1 \rightarrow e_2 \ldots \rightarrow e_n \) such that
  - each step \( e_i \rightarrow e_{i+1} \) is optimising in some resource domain \( R_i \)
  - and (at least) non-increasing for overall cost model \( R \)
  - Often: \( R \) is product of the \( R_i \)

- Usage of program logic: find operators \( F_1 \) and \( F_2 \) such that
  - \( \triangleright e_1 : [F_1(E, h) \leq r] \) and \( \triangleright e_2 : [r \leq F_2(E, h)] \)
  - \( \forall E \ h. \ F_2(E, h) \leq F_1(E, h) \)

  (notation \([\ldots]\) abbreviates \( \lambda E \ h \ldots \))
- Comparisons currently carried out in meta-logic
- Often, \( F_1 = F_2 \) works
Example validations

Carried out & formalised (see paper):

- Rinard’s transformation sequence
- Tail-call optimisations (method recursion vs. method-internal loop)

Other:

- Fuel tank (frequency of calls to external sensor)
- Traces of heaps/method calls/… (specifications expressed by logical formulae or security automata)
- Parameter values
Credible compilation: motivating transformation sequence

```c
i = 0; x = 1; y = 2; WHILE i < 24 {i = i + x + y; g = 2 * i}
```

- Transformation steps: standard compiler optimisations
- Functional correctness proven in “transformation” logic

<table>
<thead>
<tr>
<th>i</th>
<th>Time</th>
<th>Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>213</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>197</td>
<td>Constant propagation and constant folding</td>
</tr>
<tr>
<td>2</td>
<td>193</td>
<td>Dead assignment elimination</td>
</tr>
<tr>
<td>3</td>
<td>176</td>
<td>Branch movement, inlining, redundant test elimination</td>
</tr>
<tr>
<td>4</td>
<td>126</td>
<td>Induction variable elimination</td>
</tr>
<tr>
<td>5</td>
<td>126</td>
<td>Loop unrolling <em>without</em> code sharing</td>
</tr>
<tr>
<td>6</td>
<td>82</td>
<td>Dead code elimination</td>
</tr>
<tr>
<td>7</td>
<td>66</td>
<td>Expression folding</td>
</tr>
</tbody>
</table>

(Original loop unrolling shares code \(\sim\) additional jump instruction)
Verification strategy

- Define specification table $ST$ (i.e. annotations for methods and loops)
- Define proof context $G$ (uniform construction)
- Verify $G$ with respect to $ST$ (notation $ST \models G$)
  
  - for $(c.m(a), P) \in G$, verify that $P = ST(c, m)$ and $G \triangleright body_{c,m} : P[x \mapsto a]$
  
  - Syntax-directed rules (a la VCG, WP)
  
  - (manual) Discharge of side conditions

- Apply derived rule for mutual recursion

$$
\begin{array}{c}
ST \models G \\
(e, P) \in G
\end{array}
\implies
\emptyset \triangleright e : P
$$

(or variant with method argument adaptation)

- Data structures: representation predicates
- Inference of specification table: interpretation of type systems (derived assertions)
Static algebras

- Two usages of static cost models:
  - Syntactic metrics (code size, number of registers used, ...)
  - Approximation of dynamic costs (abstract interpretation)
- Definition: like dynamic resource algebras, but no dependency on dynamic values
- Derivation system $\Gamma \vdash e : t, s$. Similar to effect-systems, but not necessarily merge-operator in conditionals:

  $\Gamma \vdash e : \text{bool}, s_e$
  $\Gamma \vdash e_1 : t, s_1$
  $\Gamma \vdash e_2 : t, s_2$

  $\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : t, s_e + S^{\text{if}} + s_1 + s_2$

- Approximation theorem for suitably related static and dynamic algebras
Conclusion & Future work

- Verification of resource behaviour crucial for application domains like embedded systems
- Aim: reuse/generalisation of translation validation approach
- Optimisation validation difficult in general
  - variety of metrics (global, local), and analysis
  - dependence on functional properties
- Unifying basic platform (program logic, resource algebras) may help to structure/compare analysis/transformation algorithms
- Application to optimisation of heap behaviour (LFD)
- Resource verification of algorithm libraries
- Program logics for multiple executions
  - Deeper analysis of standard transformations (Benton)
  - Automation via derived resource logics for transformations
DFG project "InfoZert" @ LMU Munich

- 1 PhD-ship, 1 Research Assistantship (pre- or post-doc)
- Duration: 2 years (initially)
- Start of project: June 2006 (ASAP)
- Topic: proof-carrying code for information flow
- Grant holders: A. Knapp, M. Hofmann, L. Beringer
- Collaboration with Siemens (D. v. Oheimb)

Please apply!