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:

$$\frac{E \vdash h, e_1 \Downarrow h_1, v_1, r_1 \qquad E\langle x := v_1\rangle \vdash h_1, e_2 \Downarrow h_2, v, r_2}{E \vdash h, \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \Downarrow h_2, v, r_1 + \mathcal{R}^{\mathsf{let}} + r_2} \quad (\text{LET})$$

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") $\leadsto$ 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?

Resource algebras (formally)

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: $\mathcal{R}^{\mathsf{int}}$, $\mathcal{R}^{\mathsf{null}}$, $\mathcal{R}^{\mathsf{var}}$, $\mathcal{R}^{\mathsf{prim}}$, $\mathcal{R}^{\mathsf{new}}_{C}$, $\mathcal{R}^{\mathsf{getf}}$, $\mathcal{R}^{\mathsf{putf}}$, $\mathcal{R}^{\mathsf{comp}}$, $\mathcal{R}^{\mathsf{let}}$, $\mathcal{R}^{\mathsf{if}}$, $\mathcal{R}^{\mathsf{call}}$ and a monotone operator $\mathcal{R}^{\mathsf{meth}}_{C,m,\overline{v}} : R \to R$.

	Time	Frames	MethCnts	MethFreq^{Id}	MethGuard
$\lvert\mathcal{R}\rvert$	$\mathcal{N}$	$\mathcal{N}$	$\mathcal{MS}(Id)$	$\mathcal{N}\times\mathcal{N}$	$\{\mathrm{tt},\mathrm{ff}\}$
$\mathcal{R}_i^{\mathrm{int}}$	1	0	$\emptyset$	$(1,0)$	tt
$\mathcal{R}^{\mathrm{null}}$	1	0	$\emptyset$	$(1,0)$	tt
$\mathcal{R}_x^{\mathrm{var}}$	1	0	$\emptyset$	$(1,0)$	tt
$\mathcal{R}^{\mathrm{prim}}$	1	0	$\emptyset$	$(1,0)$	tt
$\mathcal{R}_C^{\mathrm{new}}$	3	0	$\emptyset$	$(3,0)$	tt
$\mathcal{R}_{x,t}^{\mathrm{getf}}$	2	0	$\emptyset$	$(2,0)$	tt
$\mathcal{R}_{x,y,t}^{\mathrm{putf}}$	3	0	$\emptyset$	$(3,0)$	tt
$\mathcal{R}^{\mathrm{comp}}$	0	0	$\emptyset$	$(0,0)$	tt
$\mathcal{R}_x^{\mathrm{let}}$	1	0	$\emptyset$	$(1,0)$	tt
$\mathcal{R}_x^{\mathrm{if}}$	0	0	$\emptyset$	$(0,0)$	tt
$\mathcal{R}_f^{\mathrm{call}}$	1	0	$\emptyset$	$(1,0)$	tt
$\mathcal{R}_{C,m,\overline{v}}^{\mathrm{meth}}(r)$	$\lvert\overline{v}\rvert+2+r$	$r+1$	$r\cup_{+}\{C.m\}$	$\mathrm{Freq}_{C.m,\lvert\overline{v}\rvert}(r)$	$G_{C,m}(\overline{v})\wedge r$
$0_{\mathcal{R}}$	0	0	$\emptyset$	$(0,0)$	tt
$+_{\mathcal{R}}$	$+$	max	$\cup_{+}$	$+_{\mathrm{Freq}}$	$\wedge$
$\leq_{\mathcal{R}}$	$\leq$	$\leq$	$\subseteq_{+}$	$\leq_{\mathrm{Freq}}$	$\leq_{\mathrm{Guard}}$

Program logic

Motivation: single verification of loop bodies & method bodies

Form of judgements: $G \rhd e : P$

- 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:

$$\frac{\Gamma \rhd e_1 : A \qquad \Gamma \rhd e_2 : B}{\begin{array}{lll} \Gamma & \rhd & \text{let } x = e_1 \text{ in } e_2 \\ & : & \lambda\, E\ h\ h'\ v\ r.\ \exists\, r_1\ r_2\ h_1\ w.\ \ A\ E\ h\ h_1\ w\ r_1\ \wedge\ w \neq \perp \wedge \\ & & \qquad B\ (E\langle x := w\rangle)\ h_1\ h'\ v\ r_2 \wedge \\ & & \qquad r = r_1 + \mathcal{R}^{\text{let}} + r_2 \end{array}} \quad (\text{VLET})$$

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: $\emptyset \rhd e : P$ implies $\models e : P$

Theorem

(Relative) completeness: $\models e : P$ implies $\emptyset \rhd e : P$

Proofs formalised in Isabelle/HOL.

Meta-theoretic properties

Interpretation

Partial correctness:

$$\models e : P \Longleftrightarrow$$
$$\forall\ E\ h\ h'\ v\ r.\ E \vdash h, e \Downarrow h', v, r \rightarrow P(E, h, h', v, r)$$

Theorem

Soundness: $\emptyset \rhd e : P$ implies $\models e : P$

Theorem

(Relative) completeness: $\models e : P$ implies $\emptyset \rhd e : P$

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: $\emptyset \rhd e : P$ implies $\models e : P$

Theorem

(Relative) completeness: $\models e : P$ implies $\emptyset \rhd e : P$

Proofs formalised in Isabelle/HOL.

Meta-theoretic properties

Interpretation

Partial correctness:

$$\models e : P \Longleftrightarrow$$
$$\forall\ E\ h\ h'\ v\ r.\ E \vdash h, e \Downarrow h', v, r \rightarrow P(E, h, h', v, r)$$

Theorem

Soundness: $\emptyset \rhd e : P$ implies $\models e : P$

Theorem

(Relative) completeness: $\models e : P$ implies $\emptyset \rhd e : P$

Proofs formalised in Isabelle/HOL.

Optimisations

- Single optimisation step (operational):
  $E \vdash h, e_1 \Downarrow h_1, v, r \wedge E \vdash h, e_2 \Downarrow h'_1, w, q \Longrightarrow q \leq_R r$
- Chain of optimisation steps $e_1 \longrightarrow e_2 \ldots \longrightarrow e_n$ such that
  - each step $e_i \longrightarrow e_{i+1}$ 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
  - $\rhd e_1 : [F_1(E,h) \leq r]$ and $\rhd e_2 : [r \leq F_2(E,h)]$
  - $\forall\ E\ h.\ F_2(E,h) \leq F_1(E,h)$

  (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

```
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

i	Time	Transformation
0	213	
1	197	Constant propagation and constant folding
2	193	Dead assignment elimination
3	176	Branch movement, inlining, redundant test elimination
4	126	Induction variable elimination
5	126	Loop unrolling *without* code sharing
6	82	Dead code elimination
7	66	Expression folding

(Original loop unrolling shares code $\rightsquigarrow$ 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 $G \rhd body_{c,m} : P[x \mapsto a]$
  - Syntax-directed rules (a la VCG, WP)
  - (manual) Discharge of side conditions
- Apply derived rule for mutual recursion
  $$\frac{ST \models G \quad (e, P) \in G}{\emptyset \rhd e : P}$$
  (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:

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

- 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

Job advert

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!