Axiomatic Semantics for Java$^{\ell ight}$

David von Oheimb[1], TU München

`http://www.in.tum.de/~oheimb/`

[1] Research funded by the DFG Project Bali

Overview

- State of the Art
- Basics of Java$^{\ell ight}$
- Hoare Logic Extensions
  - Assertions
  - Validity
- Axiomatic Semantics
  - Exceptions
  - Result Value Passing
  - Dynamic Binding
  - Class Initialization
  - Soundness & Completeness
- Example
- Conclusion

State of the Art

Motivation

Object-oriented languages like Java widely used
Demand for verification (e.g. Smart Cards)
Classical approach: Hoare Logic
New challenges (e.g. dynamic binding)

Aims

- Soundness
- Usability
- Language Coverage
- Completeness

Related Work

- Arnd Poetzsch-Heffter et al.
- Bart Jacobs et al.
- Martin Wirsing et al.
- . . .

Basics of Java$^{\ell ight}$

Features

- static overloading and dynamic binding
- references to dynamically allocated objects
- full exception handling
- static fields and methods
- static initialization of classes
- *no* multithreading

$\leadsto$ similar to Java Card

Operational Semantics as before, see e.g.
`http://isabelle.in.tum.de/Bali/src/Bali4/`

Program $\Gamma :: \mathit{prog}$ is a list of classes and interfaces
State $\sigma :: \mathit{state}$ consists of global and local variables

datatype $\mathit{st} = \mathrm{st}\ (\mathit{globs})\ (\mathit{locals})$
types $\mathit{state} = \mathit{xcpt\ option} \times \mathit{st}$

normal $\sigma \equiv$ fst $\sigma =$ None

Assertions

Shallow embedding into HOL: Predicates on the state
$\leadsto$ trivial expressiveness, relative completeness (in the sense of Cook)

Auxiliary variables for relating pre- and post-states, like in $\{X{=}Z\}\ cmd\ \{X{=}Z\}$
avoiding rules of adaptation

Result stack of anonymous intermediate results (instead of meta variables)
used for handling side-effecting expressions first-class

Assertion type $\alpha\ assn = res\ list \times state \to \alpha \to bool$

Peeking at the state, e.g. *"no exception"*: $\lambda(Y{,}\sigma)\ Z.$ normal σ
assertion transformer $P \wedge\!.\ p \ \equiv \lambda(Y{,}\sigma)\ Z.\ P\ (Y{,}\sigma)\ Z \wedge p\ \sigma$
faciliates abbreviation Normal $P \equiv P \wedge\!.$ normal

Validity

Extended Hoare triples $prog \vdash \{\alpha\ assn\}\ terms \succ\ \{\alpha\ assn\}$

(Mutual) recursion not covered here (simplification for didactical purpose)

Validity of a triple (defining *partial* correctness)

$$\Gamma \models \{P\}\ t\succ\ \{Q\} \equiv \forall Y\ \sigma\ Z.\ \ P\ (Y,\sigma)\ Z\ \longrightarrow\ \text{type\_ok}\ \Gamma\ t\ \sigma \longrightarrow$$
$$\forall v\ \sigma'.\ \ \Gamma \vdash \sigma\ -t\succ\rightarrow\ (v,\sigma')\ \longrightarrow\ Q\ (\text{res}\ t\ v\ \ Y,\sigma')\ Z$$

where

- $\text{type\_ok}\ \Gamma\ t\ \sigma$ means well-typedness and conformance
- $\Gamma \vdash \sigma\ -t\succ\rightarrow\ (v,\sigma')$ is (operational) evaluation judgment
- $\text{res}\ t\ v\ \ Y\ \equiv$ if is_stmt t then Y else Res $v : Y$ pushes the result v

Exceptions

Exception information belongs to the state $\leadsto$ is visible in assertions

Normal rule scheme

$$Any_t \; \frac{\Gamma\vdash\{\mathsf{Normal}\ P1\}\ t1\succ\ \{Q1\} \quad \ldots}{\Gamma\vdash\{\mathsf{Normal}\ P\}\ t\succ\ \{Q\}}$$

Exception propagation uniform for all terms

$$Xcpt \; \frac{}{\Gamma\vdash\{(\lambda(Y,\sigma).\ P\ (\mathsf{res}\ t\ (\mathsf{arbitrary3}\ t)\ \ Y,\sigma))\ \wedge.\ \mathsf{Not}\circ\mathsf{normal}\}\ t\succ\ \{P\}}$$

Exception handling rules for `throw`, `try _ catch _`, `finally`

Result Value Passing

Result values $res = \mathsf{Res}\ (vals) \mid \mathsf{Xcpt}\ (xcpt\ option) \mid \mathsf{Lcls}\ (locals) \mid \mathsf{DynT}\ (tname)$

pushing a result onto the stack: $P{\uparrow}{:}\mathsf{v} \;\equiv \lambda(\mathsf{Y},\sigma).\ \mathsf{P}\ (\mathsf{v}:\mathsf{Y},\sigma)$

Result generation analogous to classical assignment rule, e.g.

$$Lit\ \frac{}{\Gamma\vdash\{\mathsf{Normal}\ (P{\uparrow}{:}\mathsf{Val\ v})\}\ \mathsf{Lit\ v}{-}{\succ}\ \{\mathsf{P}\}}$$

Result consumption using stack patterns, e.g.

$$AVar\ \frac{\begin{array}{c}\Gamma\vdash\{\mathsf{Normal}\ P\}\ e_1{-}{\succ}\ \{Q\}\\ \Gamma\vdash\{Q\}\ e_2{-}{\succ}\ \{\lambda(\mathsf{Val}\ i{:}\mathsf{Val}\ a{:}Y,(x,s)).\ \mathsf{let}\ (v,x') = \ldots\ \mathsf{in}\ (R{\uparrow}{:}\mathsf{Var\ v})\ (\mathsf{Y},(\mathsf{x}',\mathsf{s}))\}\end{array}}{\Gamma\vdash\{\mathsf{Normal}\ P\}\ e_1[e_2]{=}{\succ}\ \{R\}}$$

Dynamic Binding

Dependency problem: code depends on dynamically computed value D

- D passed as a special result value
- universally quantified value bound in the precondition to D

Range problem: D should be limited to be a subtype of the type rt

- restriction on D supplied to the user as the precondition $\Gamma\vdash mode{\rightarrow}D{\preceq}rt$
- meta-level proof of correctness within soundness proof
- $\leadsto$ type-safety plays a crucial role

$$\frac{\begin{array}{c}\Gamma\vdash\{\mathsf{Normal}\ P\}\ e{-}{\succ}\ \{Q\}\\ \Gamma\vdash\{Q\}\ args\doteq{\succ}\ \{\lambda\mathsf{Vals}\ vs{:}\mathsf{Val}\ a{:}.\ \ldots R{\uparrow}{:}\mathsf{DynT\ D}.\ldots\}\\ \forall D.\ \Gamma\vdash\{R{\uparrow}{:}\mathsf{DynT\ D}\ \wedge.\lambda\sigma.\ \mathsf{normal}\ \sigma\longrightarrow\Gamma\vdash\mathsf{mode}{\rightarrow}\mathsf{D}{\preceq}\mathsf{rt}\}\ \mathsf{Body\ D\ (mn,pTs)}{-}{\succ}\ \{\ldots\}\end{array}}{\Gamma\vdash\{\mathsf{Normal}\ P\}\ \{rt,\tau,mode\}e.mn(\{pTs\}args){-}{\succ}\ \{S\}}$$

Class Initialization

Triggered on demand: Initialization not syntax-driven

- potential initialization as special statements init C at all required places

Dependence on class hierarchy:

- handled by recursive calls for initialization

$$Done \; \frac{}{\Gamma \vdash \{\mathsf{Normal}\ (P \wedge\!. \ \mathsf{initd}\ C)\}\ \mathsf{init}\ C\ \{P\}}$$

$$Init \; \frac{\begin{array}{c}\text{the (class } \Gamma\ C) = (sc,\_,\_,\_,ini) \qquad sup = \text{if } C{=}\texttt{Object} \text{ then Skip else init } sc \\ \Gamma \vdash \{\mathsf{Normal}\ ((P \wedge\!. \ \mathsf{Not} \circ \mathsf{initd}\ C)\ldots)\}\ sup\ \{Q\ldots\} \\ \Gamma \vdash \{Q\ldots\}\ ini\ \{\ldots R\ldots\}\end{array}}{\Gamma \vdash \{\mathsf{Normal}\ (P \wedge\!. \ \mathsf{Not} \circ \mathsf{initd}\ C)\}\ \mathsf{init}\ C\ \{R\}}$$

Soundness

Theorem wf_prog $\Gamma \;\longrightarrow\; \Gamma \vdash \{P\}\ t\succ\ \{Q\} \longrightarrow \Gamma \models \{P\}\ t\succ\ \{Q\}$

Proof by rule induction on the derivation of triples

- exploiting the precondition type_ok of validity together with type-safety:
  wf_prog $\Gamma \;\longrightarrow\; \Gamma \vdash \sigma\ -c\rightarrow\ \sigma' \;\longrightarrow\; (\Gamma,\Lambda)\vdash c{::}\surd \;\longrightarrow\; \sigma{::}\preceq(\Gamma,\Lambda) \;\longrightarrow\; \sigma'{::}\preceq(\Gamma,\Lambda)$
- for method calls and loops: auxiliary induction on the recursive depth

Completeness

Theorem $\mathsf{wf\_prog}\ \Gamma\ \longrightarrow\ \Gamma{\models}\{P\}\ t{\succ}\ \{Q\}\ \longrightarrow\ \Gamma{\vdash}\{P\}\ t{\succ}\ \{Q\}$

Proof by structural induction with the MGF approach

$$\{\lambda Y\sigma\ Z.\ Y\sigma{=}Z\}\ t{\succ}\ \{\lambda(Y',\sigma')\ (Y,\sigma).\ \exists v.\ Y'{=}\mathsf{res}\ t\ v\ Y \wedge \Gamma{\vdash}\sigma\ {-}t{\succ}{\rightarrow}\ (v,\sigma')\}$$

- with auxiliary induction on the number of classes already initialized
- with outer induction on the number of methods already verified
- requiring the rule $hazard\ \dfrac{}{\Gamma{\vdash}\{P\ \wedge.\ \mathsf{Not} \circ \mathsf{type\_ok}\ \Gamma\ t\}\ t{\succ}\ \{Q\}}$

  for exploiting validity and to ensure finiteness of the number of methods

Example (I)

```
class Base {
  boolean vee;
  Base foo(Base x) {
    return x;
  }
}

class Ext extends Base{
  int vee;
  Ext foo(Base x) {
    ((Ext)x).vee = 1;
    return null;
  }
}

Base e=new Ext();
try {e.foo(null); }
catch (NullPointerException z) {throw z; }
```

Example (II)

enough_mem $\longrightarrow$ $tprg \vdash \{\text{Normal } Any\}\ tblk\ \{Any \wedge.\ \lambda s.\ tprg,s \vdash \text{catch SXcpt NP}\}$

where $Any = \lambda(Y,\sigma)\ Z.\ \text{True}$

$tprg = ([\,],[(Base,BaseCl),(Ext,ExtCl)]@\text{system\_classes})$

$tblk = \text{Expr}(\text{LVar } e\texttt{:=new } Ext)\texttt{;}$

$\texttt{try Expr}(\{\text{ClassT } Base,\text{ClassT } Base,\text{IntVir}\}$

$\text{Acc }(\text{LVar } e)\texttt{.}foo(\{[\text{Class } Base]\}[\text{Lit Null}]))$

$\texttt{catch}((\text{SXcpt } \texttt{NullPointer})\ z)\ (\texttt{throw}\ (\text{Acc }(\text{LVar } z)))$

- typical "backward"-style
- $\approx$ 80 steps (rule application, simplification, classical reasoning)
- $\approx$ 30% dealing with class initialization
- 4 $\times$ rule of consequence
- 4 explicit instantiations of schematic assertion variables

Conclusion

Achievements

- First sound & (relatively) complete
  Hoare logic for a Java-style language
- Insights into the role of type-safety
- One of the major applications of Isabelle/HOL

Inherent complexity leads to . . .

- extensions of the classical Hoare calculus
- challenging proof of completeness
- tedious program verification

Theorem prover . . .

- enforces fully formal treatment.
- assists with all the details

Axiomatic Semantics, applied to OO . . .

- retains its typical flavor
- is at least feasible