Hoare Logic for Java in Isabelle/HOL

David von Oheimb[1], TU München

http://www4.in.tum.de/~oheimb/

[1]Research funded by the DFG Project BALI

Overview

- State of the Art
- Java$^{\ell ight}$
- Hoare Logic Extensions
  - Assertions
  - Validity
  - Rule of Consequence
- Axiomatic Semantics for Java
  - Exceptions
  - Value Passing
  - Dynamic Binding
  - Class Initialization
- Meta-level Proofs
  - Soundness
  - Completeness
- Verification Example
- Conclusions

State of the Art (I)

Motivation

Wide-spread use of OO languages like Java
Demand for verification (*e.g.* Smart Cards)

Aims

Hoare Logic for Java
Meta-theory in Isabelle/HOL

Goals

- Soundness
- Usability
- Language Coverage
- Completeness

State of the Art (II)

Challenges Object Orientation, *e.g.* dynamic binding
Complexity of large language embedding

Related Work

America & de Boer	SPOOL
Abadi & Leino	core language
Rustan Leino *et al.*	ESC/Java
Poetzsch-Heffter & Müller	JIVE
Bart Jacobs *et al.*	LOOP
Gary Leavens *et al.*	JML
...	

Features of Java$^{\ell ight}$

Standard (should be present):

- classes and interfaces, subtyping
- dynamic object reference structures
- static overloading and dynamic binding
- full exception handling

Extras:

- static fields and methods
- static class initialization

***not* included:**

- multi-threading
- access modifiers and packages

$\leadsto$ almost Java Card

Assertions

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

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

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

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

Peeking at the state: Normal $P \equiv \lambda Y\ \sigma\ Z.\ P\ Y\ \sigma\ Z \wedge \mathrm{fst}\ \sigma = \mathrm{None}$

Substituting a result: $P{\leftarrow}v\ \equiv \lambda Y\ \sigma\ Z.\ P\ v\ \sigma\ Z$

Validity

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

(Mutual) recursion supported with triples as assumptions
not mentioned here (simplification for didactic purposes)

Validity of a triple for partial correctness

$$\Gamma\models\{P\}\ t\ \{Q\} \equiv \forall Y\ \sigma\ Z.\ \ P\ Y\ \sigma\ Z \longrightarrow \mathsf{type\_ok}\ \Gamma\ t\ \sigma \longrightarrow \\ \forall v\ \sigma'.\ \Gamma\vdash\sigma \xrightarrow{t} (v,\sigma') \longrightarrow Q\ v\ \sigma'\ Z$$

where

- $\mathsf{type\_ok}\ \Gamma\ t\ \sigma \equiv$ means t is well-typed and σ conforms to environment
- $\Gamma\vdash\sigma \xrightarrow{t} (v,\sigma')$ is operational evaluation judgment

Rule of Consequence

$$conseq\ \frac{\begin{array}{r}\forall Y\ \sigma\ Z.\ P\ Y\ \sigma\ Z \longrightarrow (\exists P'\ Q'.\ \Gamma\vdash\{P'\}\ t\ \{Q'\} \wedge \\ (\forall Y'\ \sigma'.\ \exists Y\ Z'.\ (P'\ Y\ \sigma\ Z' \longrightarrow Q'\ Y'\ \sigma'\ Z') \longrightarrow Q\ Y'\ \sigma'\ Z))\end{array}}{\Gamma\vdash\{P\}\ t\ \{Q\}}$$

- strongest imaginable form, others are derivable
- gives adaptation completeness:

$$(\forall t.\ \Gamma\models\{P\}\ t\ \{Q\} \longrightarrow \Gamma\models\{P'\}\ t\ \{Q'\}) \longrightarrow$$
$$(\forall t.\ \Gamma\vdash\{P\}\ t\ \{Q\} \longrightarrow \Gamma\vdash\{P'\}\ t\ \{Q'\})$$

- enables extrusion of precondition and pre-state:

$$\frac{\forall Y\ \sigma\ Z.\ P\ Y\ \sigma\ Z \longrightarrow \Gamma\vdash\{\lambda Y'\ \sigma'\ Z'.\ (Y',\sigma') = (Y,\sigma)\}\ t\ \{\lambda Y\ \sigma\ Z'.\ Q\ Y\ \sigma\ Z\}}{\Gamma\vdash\{P\}\ t\ \{Q\}}$$

Exceptions

Simple representation as a state component $\leadsto$ visible in assertions

Alternatives: different postconditions or different triples

Normal rules *e.g.*

$$Seq\ \frac{\Gamma \vdash \{\mathsf{Normal}\ P\}\ c_1\ \{Q\} \quad \Gamma \vdash \{Q\}\ c_2\ \{R\}}{\Gamma \vdash \{\mathsf{Normal}\ P\}\ c_1\,;c_2\ \{R\}}$$

Exception propagation uniform for all terms

$$Xcpt\ \frac{}{\Gamma \vdash \{P \leftarrow \ldots \wedge.\ \mathsf{Not} \circ \mathsf{normal}\}\ t\ \{P\}}$$

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

Value Passing

State Extrusion making the state available in the whole triple, *e.g.*

$$\forall l.\ \Gamma \vdash \{Q \wedge\!.\ (\lambda\sigma.\ l = \mathsf{locals}\ (\mathsf{snd}\ \sigma))\ ;.\ \mathsf{set\_lvars}\ new\}\ t\ \{\mathsf{set\_lvars}\ l\ .;\ R\}$$
$$\equiv \Gamma \vdash \{Q\ ;\mathsf{tmp\_lvars}\ new\}\ t\ \{R\}$$

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

$$Lit\ \frac{}{\Gamma \vdash \{\mathsf{Normal}\ (P \leftarrow \mathsf{Val}\ v)\}\ \mathsf{Lit}\ v\ \{P\}}$$

Result Passing and Extrusion *e.g.*

$$AVar\ \frac{\Gamma \vdash \{\mathsf{Normal}\ P\}\ e_1\ \{Q\} \qquad \forall a.\ \Gamma \vdash \{Q \leftarrow \mathsf{Val}\ a\}\ e_2\ \{\lambda \mathsf{Val}\ i\ \sigma\ Z.\ \mathsf{let}\ (v,\sigma') = f\ i\ a\ \sigma\ \mathsf{in}\ R\ (\mathsf{Var}\ v)\ \sigma'\ Z\}}{\Gamma \vdash \{\mathsf{Normal}\ P\}\ e_1[e_2]\ \{R\}}$$

Dynamic Binding

Dependency problem: code depends on dynamically computed value D

- Solution: state extrusion for D
- Alternative: using '`this`' pointer and special rules enumerating D

Range problem: D should be limited to be subtype of τ

$\leadsto$ type safety plays crucial role

- Solution: Restriction on D supplied as precondition $\Gamma\vdash mode{\rightarrow}D{\preceq}\tau$; Justification by meta-level proof within soundness proof
- Alternative: Leave proof obligation to the user

$$Call\ \frac{\begin{array}{c}\Gamma\vdash\{\mathsf{Normal}\ P\}\ e\ \{Q\} \quad \forall a.\ \Gamma\vdash\{Q{\leftarrow}\mathsf{Val}\ a\}\ args\ \{R\ a\}\\ \forall a\ vs\ D.\ \Gamma\vdash\{(R\ a{\leftarrow}\mathsf{Vals}\ vs \wedge.\ (\lambda\sigma.\ \ldots\ D = \mathsf{target}\ mode\ (\mathsf{snd}\ \sigma)\ a\ dC\\ \wedge\ \ldots\ \Gamma\vdash mode{\rightarrow}D{\preceq}\tau))\ ;\mathsf{tmp\_lvars}\ \ldots\}\ \mathsf{Body}\ D\ (mn,pTs)\ \{S\}\end{array}}{\Gamma\vdash\{\mathsf{Normal}\ P\}\ \{\tau,dC,mode\}e..mn(\{pTs\}args)\ \{S\}}$$

Class Initialization

Not syntax-driven but triggered on demand

- potential initialization as special statements: init C (where required)

Dependency on class hierarchy

- recursive calls for initialization

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

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

Soundness

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

Proof by rule induction on the derivation of triples

Recursion: auxiliary induction on recursive depth

Loops: auxiliary induction on derivation of evaluation judgment

Method Calls:

exploiting type_ok as precondition induced by validity

plus type soundness

Completeness

Theorem $\text{ws\_prog}\ \Gamma \longrightarrow \Gamma \models \{P\}\ t\ \{Q\} \ \longrightarrow\ \Gamma \vdash \{P\}\ t\ \{Q\}$

Proof with the Most General Formula (MGF) approach, *i.e.* deriving the lemma $\text{ws\_prog}\ \Gamma \longrightarrow \Gamma \vdash \{\lambda Y\ \sigma\ Z.\ \sigma = Z\}\ t\ \{\lambda v\ \sigma'\ \sigma.\ \Gamma \vdash \sigma \xrightarrow{t} (v,\sigma')\}$ by structural induction.

Finiteness crucial where structure does not reduce:

- outer auxiliary induction on the number of classes already initialized
- outermost auxiliary induction on the number of methods already verified

Deriving the theorem from the lemma is easy, but for exploiting validity requires $hazard\ \dfrac{}{\Gamma \vdash \{P \wedge.\ \text{Not} \circ \text{type\_ok}\ \Gamma\ t\}\ t\ \{Q\}}$

Example (I)

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

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


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

Example (II)

proving $\texttt{tprg}\vdash\{\mathsf{Normal}\ (\lambda Y\ \sigma\ Z.\ \mathsf{heap\_free}\ 3\ \sigma \wedge \neg\mathsf{initd}\ \texttt{Base}\ \sigma \wedge \neg\mathsf{initd}\ \texttt{Ext}\ \sigma)\}$
$\texttt{test}\ \{\lambda Y\ \sigma\ Z.\ \mathsf{fst}\ \sigma = \mathsf{Some}\ (\mathsf{StdXcpt}\ \texttt{NullPointer})\}$

where $\texttt{tprg} = ([],[\mathsf{BaseClass},\mathsf{ExtClass}]@\mathsf{standard\_classes})$
$\texttt{test} = \mathsf{Expr}(\mathsf{LVar}\ \texttt{e:=new}\ \texttt{Ext});$
$\texttt{try}\ \mathsf{Expr}(\{\texttt{CTBase},\texttt{CTBase},\mathsf{IntVir}\}!!\texttt{e..foo}(\{[\mathsf{Class}\ \texttt{Base}]\}[\mathsf{Lit}\ \mathsf{Null}]))$
$\texttt{catch}((\mathsf{SXcpt}\ \texttt{NullPointer})\ \texttt{z})\ (\texttt{throw}\ (!!\texttt{z}))$

- typical syntax-directed "backwards"-style,
  computing weakest preconditions for schematic assertion variables
- approximate proof measures:
  100 steps (40 rule applications, simplification, classical reasoning)
  30% dealing with class initialization
  10 applications of rule of consequence
  10 explicit instantiations of schematic assertion variables

Achievements

- Machine-checked sound Hoare Logic for Java sublanguage
- First (relative) complete Hoare Logic for OO language
- Extensions to classical Hoare logic:
  - simple exception handling
  - stronger rule of consequence
  - state extrusion, dependent code
  - result value handling
- Insights into the role of type safety
- A major application of Isabelle/HOL

Experience

Inherent language complexity leads to . . .

- extensions of classical Hoare calculus
- challenging proof of completeness
- tedious but feasible program verification
- indispensable support by theorem prover

Theorem prover . . .

- enforces fully formal treatment
- gives maximal confidence in results
- assists with all the details

Hoare logic for OO . . .

- retains its typical flavor
- is feasible — with machine support