Axiomatic Semantics for Java$^{\ell ight}$
– *Extended Abstract* –

David von Oheimb[⋆]

Technische Universität München
http://www.in.tum.de/~oheimb/

Abstract. We introduce a Hoare-style calculus for a nearly full subset of sequential Java, which we call Java$^{\ell ight}$.
This axiomatic semantics has been proved sound and complete wrt. our operational semantics of Java$^{\ell ight}$ described earlier. The proofs also give new insights into the role of type-safety. All the formalization and proofs have been done with the theorem prover Isabelle/HOL.

1 Introduction

Since languages like Java are widely used in safety-critical applications, verification of object-oriented programs has grown more and more important. A first step towards verification seems to be developing a suitable axiomatic semantics (aka "Hoare logic") for such languages.

Recently several proposals for Hoare logics for object-oriented languages, e.g. [dB99,PHM99,HJ00], have been given. They deal with some small core language and are partially proved sound (on paper), but are known to be incomplete or at least have not been proved complete. Our new logic, in part inspired by [PHM99], has the following special merits.

- Apart from static overloading and dynamic binding of methods as well as references to dynamically allocated objects, is also covers full exception handling, static fields and methods, and static initialisation of classes. Thus our sequential sublanguage Java$^{\ell ight}$ is almost the same as Java Card[Sun99].
- Instead of modeling expressions with side-effects as assignments to intermediate variables, it handles all expressions and variables first-class. Thus programs to be verified do not need to undergo an artificial structural transformation.
- It is both sound – wrt. a mature formalization of the operational semantics of Java – and complete. This means that programs using even non-trivial features like mutual recursion, dynamic binding, and static initialisation can be proved correct.
- Apart from being rigorously and unambiguously defined (in the interactive theorem proving system Isabelle/HOL[Pau94]), it has been proved sound and complete within the system. This gives maximal confidence in the results obtained.

[⋆] Research funded by the DFG Project Bali, http://isabelle.in.tum.de/Bali/

2 Some basics of the Java$^{\ell ight}$ formalization

Our axiomatic semantics inherits all features concerning type declarations and the program state from our operational semantics of Java$^{\ell ight}$. See [ON99] for a more detailed description.

Here we just recall that a program Γ (which serves as the context for most judgments) consists of a list of class and interface declarations and that the execution state is defined as

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

where *globs* and *locals* map class references to objects (including class objects) and variable names to values, respectively, and *xcpt* references an exception object. Using the projection operators on tuples, we define e.g. $\mathsf{normal}\ \sigma \equiv \mathsf{fst}\ \sigma = \mathsf{None}$, which expresses that in state σ there is no pending exception, and write $\mathsf{snd}\ \sigma$ to refer to the state without the information on exceptions, typically denoted by s.

A term of Java$^{\ell ight}$ is either an expression, a statement, a variable, or an expression list, and has a corresponding result. For uniformity, even a statement has a (dummy) result, called Unit. The result of a variable is an *lval*, which is a value (for read access) and a state update function (for write access).

types $terms = (expr + stmt) + var + expr\ list$
types $vals = val + lval + val\ list$
types $lval = val \times (val \to state \to state)$

There are many other auxiliary type and function definitions which we cannot define here for lack of space. The complete Isabelle sources, including an example, may be obtained from `http://isabelle.in.tum.de/Bali/src/Bali4/`.

3 The axiomatic semantics

3.1 Assertions

In our axiomatic semantics we shallow-embed assertions in the meta logic HOL, i.e. define them as predicates on (basically) the state, making the dependence on the state explicit and simplifying their handling within Isabelle. This general approach is extended in two ways.

- We let the assertions depend also on so-called *auxiliary variables* (denoted by the meta variable Z of any type α), which are required to relate variable contents between pre- and postconditions, as discussed in [Sch97].
- We extend the state by a stack (implemented as a list and denoted by Y) of result values of type res, which are used to transfer results between Hoare triples. In an operational semantics, these nameless values can be referred to via meta variables, but in an axiomatic semantics, this simple technique is impossible since all values in a triple are (logically) bound to that scope (by universal quantification).

As a result, we define the type of assertions (with parameter α) as

types $\alpha\ assn = \mathsf{res}\ list \times state \to \alpha \to bool$
datatype $\mathsf{res} = \mathsf{Res}\ (vals) \mid \mathsf{Xcpt}\ (xcpt\ option) \mid \mathsf{Lcls}\ (locals) \mid \mathsf{DynT}\ (tname)$

We write e.g. $\mathsf{Val}\ v$ as an abbreviation for $\mathsf{Res}\ (\mathsf{In1}\ v)$, injecting a value v into res. Names like Val and DynT are used not only as constructors, but also as (destructor) patterns. For example, $\lambda\mathsf{Val}\ v{:}Y.\ f\ v\ Y$ is a function on the result stack that expects a value v as the top element and passes it to f together with the rest of the stack, referred to by Y.

In order to keep the Hoare rules short and thus more readable, we define several assertion (predicate) transformers.

- $\lambda s :\ P\ s \equiv \lambda(Y,\sigma).\ P\ (\mathsf{snd}\ \sigma)\ (Y,\sigma)$ allows P to peek at the state.
- $P \wedge.\ p\ \equiv \lambda(Y,\sigma)\ Z.\ P\ (Y,\sigma)\ Z \wedge p\ \sigma$ means that not only P holds but also p (applied to the program state only). $\mathsf{Normal}\ P \equiv P \wedge.\ \mathsf{normal}$ is a simple application stating that P holds and no exception has occurred.
- $P {\leftarrow}{:}f \equiv \lambda(Y,\sigma).\ P\ (Y,f\ \sigma)$ means that P holds for the state transformed by f.
- $P\ ;.\ f\ \equiv \lambda(Y,\sigma)\ Z.\ \exists\sigma'.\ P\ (Y,\sigma')\ Z \wedge \sigma = f\ \sigma'$ means that P holds for some state σ and the current state is derived from σ by the state transformer f.

3.2 Hoare triples and validity

We define triples as judgments of the form $prog\vdash\{\alpha\ assn\}\ terms\succ\ \{\alpha\ assn\}$ with some obvious variants for the different sorts of terms, e.g. $\Gamma\vdash\{P\}\ e\text{-}\succ\ \{Q\} \equiv \Gamma\vdash\{P\}\ \mathsf{In1}(\mathsf{Inl}\ e)\succ\ \{Q\}$ and $\{P\}\ .c.\ \{Q\} \equiv \{P\}\ \mathsf{In1}(\mathsf{Inr}\ c)\succ\ \{Q\}$.

Here we simplify the presentation by leaving out triples as assumptions within judgments, which are necessary to handle recursion; we have discussed this issue in detail in [Ohe99]. The validity of triples is defined as

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

where Y stands for the result stack and Z denotes the auxiliary variables. The judgment $\mathsf{type\_ok}\ \Gamma\ t\ \sigma$ means that the term t is well-typed (if σ is a normal state) and that all values in σ conform to their static types. This additional precondition is required to ensure soundness, as discussed in §3.5. $\Gamma\vdash\sigma\ -t\succ\rightarrow\ (v,\sigma')$ is the evaluation judgment from the operational semantics meaning that from the initial state σ the term t evaluates to a value v and final state σ'. Note that we define partial correctness.

Unless t is statement, the result value v is pushed onto the result stack via $\mathsf{res}\ t\ v\ Y \equiv \mathsf{if\ is\_stmt}\ t\ \mathsf{then}\ Y\ \mathsf{else\ Res}\ w{:}Y$.

3.3 Result value passing

We define the following abbreviations for producing and consuming results:

- $P{\uparrow}{:}w\ \equiv \lambda(Y,\sigma).\ P\ (w{:}Y,\sigma)$ means that P holds where the result w is pushed.
- $\lambda w{:}.\ P\ w \equiv \lambda(w{:}Y,\sigma).\ P\ w\ (Y,\sigma)$ expects and pops a result w and asserts $P\ w$.

A typical application of the former is the rule for literal values v:

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

Analogously to the well-known assignment rule, it states that the postcondtion P can be derived for a literal expression (i.e., constant) v if P – with the value v inserted – holds as the precondition and the state is normal.

The rule for array variables handles result values in a more advanced way:

$$AVar\ \frac{\Gamma\vdash\{\mathsf{Normal}\ P\}\ e_1\text{-}\!\succ\ \{Q\} \qquad \Gamma\vdash\{Q\}\ e_2\text{-}\!\succ\ \{\lambda\mathsf{Val}\ i\!:\!.\ \mathsf{RefVar}\ (\mathsf{avar}\ \Gamma\ i)\}}{\Gamma\vdash\{\mathsf{Normal}\ P\}\ e_1[e_2]\!=\!\succ\ \{R\}}$$

where $\mathsf{RefVar}\ vf\ P \equiv \lambda(\mathsf{Val}\ a\!:\!Y,(x,s)).\ \mathsf{let}\ (v,x') = vf\ a\ x\ s\ \mathsf{in}\ (P\!\uparrow\!:\!\mathsf{Var}\ v)\ (Y,(x',s))$ Both subexpressions are evaluated in sequence, where Q as intermediate assertion typically involves the result of e_1. The final postcondition R is modified for the proof on e_2 as follows: from the result stack two values are expected and popped, namely i (the index) and a (an address) of e_2 and e_1, respectively. Out of these and the intermediate state (x,s), the auxiliary function avar computes the variable v, which is pushed as the final result, and (possibly) an exception x'.

For terms involving a condition, we define the assertion $P'\!\uparrow\!:\!\mathsf{Bool}\!=\!b \equiv \lambda(Y,\sigma)\ Z.\exists v.\ P'\ (P\!\uparrow\!:\!\mathsf{Val}\ v)\ (Y,\sigma)\ Z \wedge (\mathsf{normal}\ \sigma \longrightarrow \mathsf{the\_Bool}\ v\!=\!b)$ expressing (basically) that the result of a preceding boolean expression is b. Together with the expression $(\mathsf{if}\ b\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2)$ depending on b and $P'\!\uparrow\!:\!\mathsf{Bool}\!=\!b$ identifying b with the result of e_0, we can describe both branches of conditional terms with a single triple, like in

$$Cond\ \frac{\Gamma \vdash\{\mathsf{Normal}\ P\}\ e_0\text{-}\!\succ\ \{P'\} \quad \forall b.\ \Gamma\vdash\{P'\!\uparrow\!:\!\mathsf{Bool}\!=\!b\}\ (\mathsf{if}\ b\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2)\text{-}\!\succ\ \{Q\}}{\Gamma\vdash\{\mathsf{Normal}\ P\}\ e_0\ \ ?\ e_1\ \ :\ e_2\text{-}\!\succ\ \{Q\}}$$

The value b is universally quantified, such that when applying this rule, one has to prove its second antecedent for any possible value, i.e., True or False. What is a notational convenience here (to avoid two triples, one for each case), will be essential for the *Call* rule, given below.

The rules for the standard statements appear almost as usual:

$$Skip\ \frac{}{\Gamma\vdash\{P\}\ .\mathsf{Skip}.\ \{P\}} \qquad Loop\ \frac{\Gamma\vdash\{P\}\ e\text{-}\!\succ\ \{P'\} \quad \Gamma\vdash\{P'\!\uparrow\!:\!\mathsf{Bool}\!=\!\mathsf{True}\}\ .c.\ \{P\}}{\Gamma\vdash\{P\}\ .\mathtt{while}(e)\ c.\ \{P'\!\uparrow\!:\!\mathsf{Bool}\!=\!\mathsf{False}\}}$$

Note that in all[1] rules (except Loop for obvious reasons) the postconditions of the conclusion is a variable. Thus in the typical "backward-proof" style of Hoare logic the rules are applied easily.

3.4 Dynamic binding

The great challenge of an axiomatic semantics for an object-oriented language is dynamic binding in method calls, for two reasons.

First, the code selected depends on the class D dynamically computed from the reference expression e. The range of values for D depends on the whole program and thus cannot be fixed locally, in contrast to the two possible boolean values appearing in conditional terms described above. Standard Hoare triples cannot express such an unbound case distinction. We handle this problem with the strong technique given above, using universal quantification and the precondition $R\!\uparrow\!:\!\mathsf{DynT}\ D \wedge\ldots$ with the special result value $\mathsf{DynT}\ D$. An alternative

[1] the rules not mentioned here may be found in the appendix

solution is given in [PHM99], where D is referred to via `this` and the possible variety of D is handled in a cascadic way using several special rules.

Second, the actual value D often can be inferred statically, but in general for invocation mode “virtual”, one can only know that it is a subtype of some reference type rt computed by static analysis during type-checking. The intuitive – but absolutely non-trivial – reason why the subtype relation $\mathsf{Class}\ D{\preceq}\mathsf{RefT}\ rt$ holds is of course type-safety. The problem here is how to establish this relation. [PHM99] simply places it into the precondition of the consequence of the appropriate rule, but in general this puts a heavy burden on the rule user, making the calculus at least practically incomplete. In contrast, our solution puts the relation (as the formula $\Gamma\vdash mode{\to}D{\preceq}rt$) into the precondition of an antecedent and thus provides the user with an additional helpful assumption, transferring the proof burden once and for all to the soundness proof.

The remaining parts of the rule for method calls deals with the unproblematic issues of argument evaluation, setting up the local variables (including parameters) of the called method and restoring the previous local variables on return, for which we use the special result value Lcls.

$$\mathit{Call}\ \frac{\begin{array}{c} \Gamma\vdash\{\mathsf{Normal}\ P\}\ e\text{-}\!\succ\ \{Q\} \\ \Gamma\vdash\{Q\}\ args\doteq\!\succ\ \{\lambda\mathsf{Vals}\ vs{:}\mathsf{Val}\ a{:}.\ \lambda s{:}\ \mathsf{let}\ D = \mathsf{dyn\_class}\ mode\ s\ a\ \tau\ \mathsf{in} \\ \quad R\uparrow{:}\mathsf{DynT}\ D\uparrow{:}\mathsf{Lcls}\ (\mathsf{locals}\ s){\leftarrow}{:}\mathsf{init\_lvars}\ \Gamma\ D\ (mn,pTs)\ mode\ a\ vs\} \\ \forall D.\ \Gamma\vdash\{R\uparrow{:}\mathsf{DynT}\ D\ \wedge.\lambda\sigma.\ \mathsf{normal}\ \sigma \longrightarrow \Gamma\vdash mode{\to}D{\preceq}rt\} \\ \mathsf{Body}\ D\ (mn,pTs)\text{-}\!\succ\ \{\lambda\mathsf{Val}\ v{:}\mathsf{Lcls}\ l{:}.\ S\uparrow{:}\mathsf{Val}\ v{\leftarrow}{:}\mathsf{set\_lvars}\ l\} \end{array}}{\Gamma\vdash\{\mathsf{Normal}\ P\}\ \{rt,\tau,mode\}e.mn(\{pTs\}args)\text{-}\!\succ\ \{S\}}$$

3.5 Soundness and completeness

With the help of Isabelle/HOL, we have proved soundness and completeness

$$\mathsf{wf\_prog}\ \Gamma \Longrightarrow \Gamma\models\{P\}\ t\!\succ\ \{Q\} = \Gamma\vdash\{P\}\ t\!\succ\ \{Q\}$$

where $\mathsf{wf\_prog}\ \Gamma$ means that the program Γ is well-formed. As usual, soundness is proved by rule induction on the derivation of triples. Surprisingly, type-safety plays a crucial role here. The important fact that for method calls the subtype relation $\mathsf{Class}\ D{\preceq}\mathsf{RefT}\ rt$ mentioned above holds can be derived only if the state conforms to the environment. This was the reason to bring the judgment $\mathsf{type\_ok}$ into our definition of validity, which also gives rise to the new rule (required for the completeness proof)

$$\mathit{hazard}\ \frac{}{\Gamma\vdash\{P\ \wedge.\ \mathsf{Not} \circ \mathsf{type\_ok}\ \Gamma\ t\}\ t\!\succ\ \{Q\}}$$

indicating that if at any time conformance was violated, anything could happen.

Completeness is proved (basically) by structural induction with the MGF approach discussed in [Ohe99]. This includes an outer auxiliary induction on the number of methods already verified, which requires well-typedness in order to ensure that for any program there is only a finite number of methods. Due to class initialisation, an extra induction on the number of classes already initialized is required.

References

[dB99] Frank de Boer. A WP-calculus for OO. In *Foundations of Software Science and Computation Structures*, volume 1578 of *LNCS*. Springer-Verlag, 1999.

[HJ00] Marieke Huisman and Bart Jacobs. Java program verfication via a Hoare logic with abrupt termination. In *Fundamental Approaches to Software Engineering (FASE'00)*, LNCS. Springer-Verlag, 2000. to appear.

[Ohe99] David von Oheimb. Hoare logic for mutual recursion and local variables. In C. Pandu Rangan, V. Raman, and R. Ramanujam, editors, *FST&TCS'99*, volume 1738 of *LNCS*, pages 168–180. Springer-Verlag, 1999.

[ON99] David von Oheimb and Tobias Nipkow. Machine-checking the Java specification: Proving type-safety. In Jim Alves-Foss, editor, *Formal Syntax and Semantics of Java*, volume 1523 of *LNCS*. Springer-Verlag, 1999. `http://isabelle.in.tum.de/Bali/papers/Springer98.html`.

[Pau94] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *LNCS*. Springer-Verlag, 1994. For an up-to-date description, see `http://isabelle.in.tum.de/`.

[PHM99] Arnd Poetzsch-Heffter and Peter Müller. A programming logic for sequential Java. In S. D. Swierstra, editor, *Programming Languages and Systems (ESOP '99)*, volume 1576 of *LNCS*, pages 162–176. Springer-Verlag, 1999.

[Sch97] Thomas Schreiber. Auxiliary variables and recursive procedures. In *TAPSOFT'97*, volume 1214 of *LNCS*, pages 697–711. Springer-Verlag, 1997.

[Sun99] Sun. *Java Card Spec.*, 1999. `http://java.sun.com/products/javacard/`.

A The remaining rules

$$\mathit{conseq}\ \frac{\begin{array}{c}\forall Y\ \sigma\ Z\,.\ P\ \ (Y\,,\sigma)\ Z\ \longrightarrow (\exists P'\ \ Q'.\ \mathit{\Gamma}\vdash\{P'\}\ t\succ\ \{Q'\}\ \wedge\ (\forall w\ \sigma'.\\ (\forall Y'\ \ Z'.\ P'\ (Y',\sigma)\ Z'\ \longrightarrow\ Q'\ (\mathsf{res}\ t\ w\ Y',\sigma')\ Z')\ \longrightarrow\ Q\ \ (\mathsf{res}\ t\ w\ Y\,,\sigma')\ Z\,))\end{array}}{\mathit{\Gamma}\vdash\{P\ \}\ t\succ\ \{Q\ \}}$$

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

$$\mathit{Super}\ \frac{}{\mathit{\Gamma}\vdash\{\mathsf{Normal}\ (\lambda s:\ P\!\uparrow\!:\mathsf{Val}\ (\mathsf{this}\ s))\}\ \mathtt{super}\text{-}\!\succ\ \{P\}}$$

$$\mathit{LVar}\ \frac{}{\mathit{\Gamma}\vdash\{\mathsf{Normal}\ (\lambda s:\ P\!\uparrow\!:\mathsf{Var}\ (\mathsf{lvar}\ vn\ s))\}\ \mathsf{LVar}\ vn\!=\!\succ\ \{P\}}$$

$$\mathit{FVar}\ \frac{\mathit{\Gamma}\vdash\{\mathsf{Normal}\ P\}\ .\mathsf{init}\ C.\ \{Q\}\qquad \mathit{\Gamma}\vdash\{Q\}\ e\text{-}\!\succ\ \{\mathsf{RefVar}\ (\mathsf{fvar}\ \ C\ \mathit{stat}\ \mathit{fn})\ R\}}{\mathit{\Gamma}\vdash\{\mathsf{Normal}\ P\}\ \{C,\mathit{stat}\}e.\mathit{fn}\!=\!\succ\ \{R\}}$$

$$\mathit{Acc}\ \frac{\mathit{\Gamma}\vdash\{\mathsf{Normal}\ P\}\ va\!=\!\succ\ \{\lambda\mathsf{Var}\ (v,f)\!:\!.\ \ Q\!\uparrow\!:\mathsf{Val}\ \ v\}}{\mathit{\Gamma}\vdash\{\mathsf{Normal}\ P\}\ \mathsf{Acc}\ va\text{-}\!\succ\ \{Q\}}$$

$$\mathit{Ass}\ \frac{\begin{array}{c}\mathit{\Gamma}\vdash\{\mathsf{Normal}\ P\}\ va\!=\!\succ\ \{Q\}\\ \mathit{\Gamma}\vdash\{Q\}\ \ e\text{-}\!\succ\ \{\lambda\mathsf{Val}\ v\!:\!\mathsf{Var}\ (w,f)\!:\!.\ \ R\!\uparrow\!:\mathsf{Val}\ v\!\leftarrow\!:\mathsf{assign}\ f\ v\}\end{array}}{\mathit{\Gamma}\vdash\{\mathsf{Normal}\ P\}\ va\mathtt{:=}e\text{-}\!\succ\ \{R\}}$$

$$\mathit{Nil}\ \frac{}{\{\mathsf{Normal}\ \ P\!\uparrow\!:\mathsf{Vals}\ []\}\ []\doteq\!\succ\ \{P\}}$$

$$\mathit{Cons}\ \frac{\Gamma\vdash\{\mathsf{Normal}\ P\}\ e\text{-}\!\succ\ \{Q\}\qquad \Gamma\vdash\{Q\}\ es\doteq\!\succ\ \{\lambda\mathsf{Vals}\ vs{:}\mathsf{Val}\ v{:}.\ R{\uparrow}{:}\mathsf{Vals}\ (v{:}vs)\}}{\Gamma\vdash\{\mathsf{Normal}\ P\}\ e{:}es\doteq\!\succ\ \{R\}}$$

$$\mathit{NewC}\ \frac{\Gamma\vdash\{\mathsf{Normal}\ P\}\ .\mathsf{init}\ C.\ \{\mathsf{Alloc}\ \Gamma\ (\mathsf{CInst}\ C)\ \mathit{id}\ Q\}}{\Gamma\vdash\{\mathsf{Normal}\ P\}\ \texttt{new}\ C\text{-}\!\succ\ \{Q\}}$$

where $\mathsf{Alloc}\ \Gamma\ \mathit{otag}\ f\ P\ \equiv$

$$\lambda(Y,(x,s))\ Z.\ \forall\sigma'\ a.\ \Gamma\vdash(f\ x,s)\ -\mathsf{halloc}\ \mathit{otag}\!\succ\! a\!\rightarrow\ \sigma'\longrightarrow(P{\uparrow}{:}\mathsf{Val}\ (\mathsf{Addr}\ a))\ (Y,\sigma')\ Z$$

$$\mathit{NewA}\ \frac{\begin{array}{c}\Gamma\vdash\{\mathsf{Normal}\ P\}\ .\mathsf{init}\ (\mathsf{if}\ (\exists C.\ T{=}\mathsf{Class}\ C)\ \mathsf{then}\ \mathsf{the\_Class}\ T\ \mathsf{else}\ \texttt{Object}).\ \{Q\}\\ \Gamma\vdash\{Q\}\ e\text{-}\!\succ\ \{\lambda\mathsf{Val}\ i{:}.\ \mathsf{Alloc}\ \Gamma\ (\mathsf{Arr}\ T\ (\mathsf{the\_Intg}\ i))\ (\mathsf{check\_neg}\ i)\ R\}\end{array}}{\Gamma\vdash\{\mathsf{Normal}\ P\}\ \texttt{new}\ T[e]\text{-}\!\succ\ \{R\}}$$

$$\mathit{Cast}\ \frac{\Gamma\vdash\{\mathsf{Normal}\ P\}\ e\text{-}\!\succ\ \{\lambda\mathsf{Val}\ v{:}.\ Q{\uparrow}{:}\mathsf{Val}\ v{\leftarrow}{:}\lambda(x,s).\ (\mathsf{raise\_if}\ (\neg\Gamma,s\vdash v\ \mathsf{fits}\ T)\ \texttt{ClassCast}\ x,s)\}}{\Gamma\vdash\{\mathsf{Normal}\ P\}\ \mathsf{Cast}\ T\ e\text{-}\!\succ\ \{Q\}}$$

$$\mathit{Inst}\ \frac{\Gamma\vdash\{\mathsf{Normal}\ P\}\ e\text{-}\!\succ\ \{\lambda\mathsf{Val}\ v{:}.\ \lambda s{:}\ (Q{\uparrow}{:}\mathsf{Val}\ (\mathsf{Bool}\ (v{\neq}\mathsf{Null}\wedge\Gamma,s\vdash v\ \mathsf{fits}\ \mathsf{RefT}\ T)))\}}{\Gamma\vdash\{\mathsf{Normal}\ P\}\ e\ \texttt{instanceof}\ T\text{-}\!\succ\ \{Q\}}$$

$$\mathit{Body}\ \frac{\begin{array}{c}\mathsf{the}\ (\mathsf{cmethd}\ \Gamma\ C\ \mathit{sig})=(\mathit{md},\ \_,\ \_,\ \mathit{blk},\ \mathsf{res})\\ \Gamma\vdash\{\mathsf{Normal}\ P\}\ .\mathsf{init}\ \mathit{md}.\ \{Q\}\qquad\Gamma\vdash\{Q\}\ .\mathit{blk}.\ \{R\}\quad\Gamma\vdash\{R\}\ \mathsf{res}\text{-}\!\succ\ \{S\}\end{array}}{\Gamma\vdash\{\mathsf{Normal}\ P\}\ \mathsf{Body}\ C\ \mathit{sig}\text{-}\!\succ\ \{S\}}$$

$$\mathit{Expr}\ \frac{\Gamma\vdash\{\mathsf{Normal}\ P\}\ e\text{-}\!\succ\ \{\lambda(w{:}Y,\sigma).\ Q\ (Y,\sigma)\}}{\Gamma\vdash\{\mathsf{Normal}\ P\}\ .\mathsf{Expr}\ e.\ \{Q\}}\qquad\mathit{Comp}\ \frac{\Gamma\vdash\{\mathsf{Normal}\ P\}\ .c_1.\ \{Q\}\quad\Gamma\vdash\{Q\}\ .c_2.\ \{R\}}{\Gamma\vdash\{\mathsf{Normal}\ P\}\ .c_1;c_2.\ \{R\}}$$

$$\mathit{If}\ \frac{\Gamma\vdash\{\mathsf{Normal}\ P\}\ e\text{-}\!\succ\ \{P'\}\qquad\forall b.\ \Gamma\vdash\{P'{\uparrow}{:}\mathsf{Bool}{=}b\}\ .(\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2).\ \{Q\}}{\Gamma\vdash\{\mathsf{Normal}\ P\}\ .\texttt{if}(e)\ c_1\ \texttt{else}\ c_2.\ \{Q\}}$$

$$\mathit{Throw}\ \frac{\Gamma\vdash\{\mathsf{Normal}\ P\}\ e\text{-}\!\succ\ \{\lambda\mathsf{Val}\ a{:}.\ Q{\leftarrow}{:}\lambda(x,s).\ (\mathsf{throw}\ a\ x,s)\}}{\Gamma\vdash\{\mathsf{Normal}\ P\}\ .\texttt{throw}\ e.\ \{Q\}}$$

$$\mathit{Try}\ \frac{\begin{array}{c}\Gamma\vdash\{\mathsf{Normal}\ P\}\ .c_1.\ \{Q\}\\ \Gamma\vdash\{(Q\ \wedge.\lambda\sigma.\ \Gamma,\sigma\vdash\mathsf{catch}\ C)\ ;.\ \mathsf{new\_xcpt\_var}\ \mathit{vn}\}\ .c_2.\ \{R\}\\ \Gamma\vdash\{Q\ \wedge.\lambda\sigma.\ \neg\Gamma,\sigma\vdash\mathsf{catch}\ C\}\ .\mathsf{Skip}.\ \{R\}\end{array}}{\Gamma\vdash\{\mathsf{Normal}\ P\}\ .\texttt{try}\ c_1\ \texttt{catch}(C\ \mathit{vn})\ c_2.\ \{R\}}$$

$$\mathit{Fin}\ \frac{\begin{array}{c}\Gamma\vdash\{\mathsf{Normal}\ P\}.c_1.\{\lambda(Y,(x,s)).\ (Q{\uparrow}{:}\mathsf{Xcpt}\ x)\ (Y,(\mathsf{None},s))\}\\ \Gamma\vdash\{\mathsf{Normal}\ Q\}.c_2.\{\lambda\mathsf{Xcpt}\ x'{:}.\ R{\leftarrow}{:}\lambda(x,s).\ (\mathsf{xcpt\_if}\ (x'{\neq}\mathsf{None})\ x'\ x,s)\}\end{array}}{\Gamma\vdash\{\mathsf{Normal}\ P\}\ .c_1\ \texttt{finally}\ c_2.\ \{R\}}$$

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

$$\mathit{Init}\ \frac{\begin{array}{c}\mathsf{the}\ (\mathsf{class}\ \Gamma\ C)=(s,\_,\_,\_,c)\qquad\mathit{sup}=\mathsf{case}\ s\ \mathsf{of}\ \mathsf{None}\rightarrow\mathsf{Skip}\mid\mathsf{Some}\ sc\rightarrow\mathsf{init}\ sc\\ \Gamma\vdash\{\mathsf{Normal}\ ((P\ \wedge.\ \mathsf{Not}\circ\mathsf{initd}\ C)\ ;.\ \mathsf{supd}\ (\mathsf{new\_stat\_obj}\ \Gamma\ C))\}\ .\mathit{sup}.\ \{Q{\uparrow}{:}.\lambda s.\ \mathsf{Lcls}\ (\mathsf{locals}\ s)\}\\ \Gamma\vdash\{Q\ ;.\ \mathsf{set\_lvars\ empty}\}\ .c.\ \{\lambda\mathsf{Lcls}\ l{:}.\ R{\leftarrow}{:}\mathsf{set\_lvars}\ l\}\end{array}}{\Gamma\vdash\{\mathsf{Normal}\ (P\ \wedge.\ \mathsf{Not}\circ\mathsf{initd}\ C)\}\ .\mathsf{init}\ C.\ \{R\}}$$