Specification of procedure quickSort

```
quickSort_spec:
  "∀σ Ps. Γ⊢
     { σ. List ´p ´next Ps }
        ´p :== PROC quickSort(´p)
     { (∃ sortedPs. List ´p ´next sortedPs ∧
        sorted (op ≤) (map  ^σcont sortedPs) ∧
        Ps <~~> sortedPs ) ∧
        (∀x. x∉set Ps ⟶ ´next x =  ^σnext x) }"

quickSort_modifies:
  "∀σ. Γ⊢
     { σ }
        ´p :== PROC quickSort(´p)
     { t. t may_only_modify_globals σ in [next] }"
```

Discussion of embedding approach

- Advantages:
  - powerful specification language and reasoning support
  - flexible for experimenting with languages
  - meta-logical aspects can be handled
- Disadvantages:
  - handling (realistic) programs can be a bit cumbersome
  - realizing convenient IDEs expensive

General approach 2

- Develop a more user-friendly language definition and logic
- Develop a dedicated verification front-end
- Use the HOL proof system as a back-end

Example: Logic for Java-KE programs

Java-KE: A Java kernel language with exceptions

A sequential object-oriented programming language:

- mutually recursive classes and interfaces
- subtyping and inheritance
- dynamic method binding
- casts
- simple exception handling mechanism

Goals and background material

Main goals:

- verification of object-oriented programs
- syntactical representation of the logic
- detailed verification in the presence of heaps

As background material see technical report.

Identifiers and basic types

We assume the following types:

- *FieldId* for field identifiers,
- *MethodId* for method identifiers,
- *CTypeId* for class type identifiers,
- *ITypeId* for interface type identifiers,
- *VarId* for variable identifiers including special identifiers `this`, `par`, `res`, and `exc` (see below);
- *Int* for integers with constants $\ldots, -1I, 0I, 1I, \ldots$ and
- *Bool* for the booleans with constants TRUE and FALSE; finally,
- *UnOp* for a suitable set of unary operators and
- *BinOp* for a suitable set of binary operators

Declarations and types

Program	=	list of *TypeDecl*
TypeDecl	=	*classDecl*(*CTypeId CTypeId ITypeIdList CBody*)
	|	*interfaceDecl*(*ITypeId ITypeIdList IBody*)
CBody	=	list of *MemberDecl*
MemberDecl	=	*fieldDecl*(*Type FieldId*)
	|	*methodDecl*(*MethodSig Statement*)
IBody	=	list of *MethodSig*
MethodSig	=	*sig*(*Type MethodId Type*)
Type	=	*booleanT* | *intT* | *nullT*
	|	*ct*(*CTypeId*) | *it*(*ITypeId*)

Statements and expressions

$$
\begin{array}{lcl}
\mathit{Statement} & = & \mathit{block}(\ \mathit{Type}\ \ \mathit{VarId}\ \ \mathit{Statement}\ ) \\
& | & \mathit{cassign}(\ \mathit{VarId}\ \ \mathit{Type}\ \ \mathit{Exp}\ ) \\
& | & \mathit{fread}(\ \mathit{VarId}\ \ \mathit{VarId}\ \ \mathit{FieldId}\ ) \\
& | & \mathit{fwrite}(\ \mathit{VarId}\ \ \mathit{FieldId}\ \ \mathit{VarId}\ ) \\
& | & \mathit{new}(\ \mathit{VarId}\ \ \mathit{CTypeId}\ ) \\
& | & \mathit{seq}(\ \mathit{Statement}\ \ \mathit{Statement}\ ) \\
& | & \mathit{if}(\ \mathit{Exp}\ \ \mathit{Statement}\ \ \mathit{Statement}\ ) \\
& | & \mathit{while}(\ \mathit{Exp}\ \ \mathit{Statement}\ ) \\
& | & \mathit{catch}(\ \mathit{Statement}\ \ \mathit{CTypeId}\ \ \mathit{VarId}\ \ \mathit{Statement}\ ) \\
& | & \mathit{invoc}(\ \mathit{VarId}\ \ \mathit{VarId}\ \ \mathit{MethodId}\ \ \mathit{Exp}\ ) \\
& | & \mathit{call}(\ \mathit{VarId}\ \ \mathit{CTypeId}\ \ \mathit{MethodId}\ \ \mathit{Exp}\ )
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{Exp} & = & \mathit{ic}(\ \mathit{Int}\ ) \mid \mathit{bc}(\ \mathit{Bool}\ ) \mid \mathit{nullc} \mid \mathit{id}(\ \mathit{VarId}\ ) \\
& | & \mathit{unary}(\ \mathit{UnOp}\ \ \mathit{Exp}\ ) \mid \mathit{binary}(\ \mathit{Exp}\ \ \mathit{BinOp}\ \ \mathit{Exp}\ )
\end{array}
$$

Context conditions

Context conditions of Java-KE:

- the context conditions of Java
- every Java-KE program contains classes `Object`, `CastExc` and `NullPExc`

Subtype relation, identifiers for dynamic entities, values

In the following, we assume that a Java-KE program Π is given

$$\preceq : \mathit{Type} \times \mathit{Type} \to \mathit{Bool}$$

We assume the types:

- *ObjId* for object identifiers
- *InstVar* for instance variables

$$\begin{array}{lcl} \mathit{Value} & = & b(\ \mathit{Bool}\ ) \\ & | & i(\ \mathit{Int}\ ) \\ & | & \mathit{null} \\ & | & \mathit{ref}(\ \mathit{CTypeId}\ \mathit{ObjId}\ ) \end{array}$$

Notations and program parts

In formulas, we write:

- `true` and `false` instead of $b(\mathrm{TRUE})$ and $b(\mathrm{FALSE})$
- $-1,\ 0,\ 1$ instead of $i(-1_I),\ i(0_I),\ i(1_I)$

We assume the types:

- *DeclMethId* for all method declarations in Π, denoted by strings of the form *cid*@*mid* with *cid* in *CTypeId* and *mid* ∈ *MethodId*
- *VirtMethId* for the so-called *virtual methods*, denoted by *tid*:*mid*, if *mid* is is a method of *tid*
- *StmtOcc* the type of all statement occurrences in Π
- *ProgPart* = *DeclMethId* ∪ *VirtMethId*∪ *StmtOcc*

Helper functions

$$
\begin{array}{lll}
\mathit{vis} & : \mathit{ProgPart} & \to 2^{\mathit{VarId}} \\
\tau & : \mathit{Value} & \to \mathit{Type} \\
\mathit{init} & : \mathit{Type} & \to \mathit{Value} \\
\mathit{styp} & : \mathit{InstVar} & \to \mathit{Type} \\
\mathit{styp} & : \mathit{VarId} \times \mathit{ProgPart} & \to \mathit{Type} \\
\mathit{rtyp} & : \mathit{ProgPart} & \to \mathit{Type} \\
\mathit{defdm} & : \mathit{Type} \times \mathit{MethodId} & \to \mathit{Bool} \\
\mathit{dm} & : \mathit{Type} \times \mathit{MethodId} & \to \mathit{DeclMethId} \\
\mathit{defvm} & : \mathit{Type} \times \mathit{MethodId} & \to \mathit{Bool} \\
\mathit{vm} & : \mathit{Type} \times \mathit{MethodId} & \to \mathit{VirtMethId} \\
\mathit{body} & : \mathit{DeclMethId} & \to \mathit{StmtOcc} \\
\_.\_ & : \mathit{Value} \times \mathit{FieldId} & \to \mathit{InstVar}
\end{array}
$$

Object store

Formalization of the object store (or heap):

- $\_(\_) : \mathit{Store} \times \mathit{InstVar} \to \mathit{Value}$ where $OS(x.f)$ yields the value of instance variable $x.f$ in state OS of the object store.
- $\_\langle \_ := \_ \rangle : \mathit{Store} \times \mathit{InstVar} \times \mathit{Value} \to \mathit{Store}$ where $OS\langle x.f := v \rangle$ yields the state of the object store after updating $x.f$ in OS by v.
- $\_\langle \_ \rangle : \mathit{Store} \times \mathit{CTypeId} \to \mathit{Store}$ where $OS\langle TID \rangle$ yields the state of the object store after allocating a new object of type $ct(TID)$.
- $\mathit{new} : \mathit{Store} \times \mathit{CTypeId} \to \mathit{Value}$ where $\mathit{new}(OS, TID)$ yields the object of type $ct(TID)$ that has been allocated last in OS.
- $\mathit{alive} : \mathit{Value} \times \mathit{Store} \to \mathit{Bool}$ where $\mathit{alive}(x, OS)$ yields true iff x is an allocated object or a constant value.

States

In Java-KE, we model states as pairs:

$$State \quad = \quad (VarId \to Value) \times (\{\$\} \to Store)$$

Explanations:

- \$ is a global variable of type *Store*
- Applications: $S(v)$ for $v \in VarId$ and $S(\$)$ otherwise
- Updates: $S[v := E]$ and $S[\$:= E]$
- Lifting: If $e \in Exp$, we write $S(e)$ for the evaluation of e in S

Operational semantics

The operational semantics inductively defines the relation *ssem*:

$$ssem \quad : \quad State \times StmtOcc \times State \to Bool$$

We write $SP : c \to SQ$ for $ssem(SP, c, SQ)$

$$\frac{\tau(S(e)) \leq T}{S : \ x = (T)e; \ \to \ S[x := S(e)]}$$

$$\frac{\tau(S(e)) \not\leq T}{S : \ x = (T)e; \ \to \ S[\$:= S(\$)\langle \texttt{CastExc} \rangle, \texttt{exc} := new(S(\$), \texttt{CastExc})]}$$

$$\frac{S(y) \neq \texttt{null}}{S : \ x = y.a; \ \to \ S[x := S(\$)(S(y).a)]}$$

Operational semantics (2)

$$\frac{S(y) = \texttt{null}}{S:\ x = y.a;\ \rightarrow\ S[\$:= S(\$)\langle \texttt{NullPExc}\rangle, \texttt{exc} := \mathit{new}(S(\$), \texttt{NullPExc})]}$$

$$\frac{S(x) \neq \texttt{null}}{S:\ x.a = e;\ \rightarrow\ S[\$:= S(\$)\langle S(x).a := S(e)\rangle]}$$

$$\frac{S(x) = \texttt{null}}{S:\ x.a = e;\ \rightarrow\ S[\$:= S(\$)\langle \texttt{NullPExc}\rangle, \texttt{exc} := \mathit{new}(S(\$), \texttt{NullPExc})]}$$

$$\frac{\mathit{true}}{S:\ x = \texttt{new}\ T();\ \rightarrow\ S[\$:= S(\$)\langle T\rangle, x := \mathit{new}(S(\$), T)]}$$

Operational semantics (3)

$$\frac{S : s_1 \rightarrow SQ,\ SQ(\texttt{exc}) = \texttt{null},\ SQ : s_2 \rightarrow SR}{S : s_1\ s_2 \rightarrow SR}$$

$$\frac{S : s_1 \rightarrow SQ,\ SQ(\texttt{exc}) \neq \texttt{null}}{S : s_1\ s_2 \rightarrow SQ}$$

$$\frac{S(e) = \texttt{true},\ S : s_1 \rightarrow SQ}{S : \texttt{if}\,(e)\{\,s_1\,\}\texttt{else}\{\,s_2\,\} \rightarrow SQ}$$

$$\frac{S(e) = \texttt{false},\ S : s_2 \rightarrow SQ}{S : \texttt{if}\,(e)\{\,s_1\,\}\texttt{else}\{\,s_2\,\} \rightarrow SQ}$$

Operational semantics (4)

$$\frac{S(e) = \texttt{false}}{S : \texttt{while}(e)\{\, s \,\} \rightarrow S}$$

$$\frac{S(e) = \texttt{true},\ S : s \rightarrow SQ,\ SQ(\texttt{exc}) = \texttt{null},\ SQ : \texttt{while}(e)\{\, s \,\} \rightarrow SR}{S : \texttt{while}(e)\{\, s \,\} \rightarrow SR}$$

$$\frac{S(e) = \texttt{true},\ S : s \rightarrow SQ,\ SQ(\texttt{exc}) \neq \texttt{null}}{S : \texttt{while}(e)\{\, s \,\} \rightarrow SQ}$$

$$\frac{S[v := \mathit{init}(T)] : s \rightarrow SQ}{S : \{\, T\ v;\ s \,\} \rightarrow SQ}$$

Operational semantics (5)

$$\frac{S:\ s_0\ \to\ SQ,\ SQ(\mathtt{exc}) = \mathtt{null}}{S:\ \mathtt{try}\{\, s_0 \,\}\mathtt{catch}(T\ v)\{\, s_1 \,\}\ \to\ SQ}$$

$$\frac{S:\ s_0\ \to\ SQ,\ SQ(\mathtt{exc}) \neq \mathtt{null},\ \tau(SQ(\mathtt{exc})) \not\leq T}{S:\ \mathtt{try}\{\, s_0 \,\}\mathtt{catch}(T\ v)\{\, s_1 \,\}\ \to\ SQ}$$

$$\frac{\begin{array}{l} S:\ s_0\ \to\ SQ,\ SQ(\mathtt{exc}) \neq \mathtt{null},\ \tau(SQ(\mathtt{exc})) \leq T, \\ SQ[v := SQ(\mathtt{exc}), \mathtt{exc} := \mathtt{null}] : s_1\ \to\ SR \end{array}}{S:\ \mathtt{try}\{\, s_0 \,\}\mathtt{catch}(T\ v)\{\, s_1 \,\}\ \to\ SR}$$

Operational semantics (6)

$$\frac{S(y) \neq \texttt{null},\ \tau(S(y)) \leq \mathit{styp}(y, \text{“}x = y.m(e);\text{”}),\ \mathit{DMI} = \mathit{dm}(\tau(S(y)), m),\ S[\texttt{this} := S(y), \texttt{par} := S(e), \texttt{res} := \mathit{init}(\mathit{rtyp}(\mathit{DMI}))] : \mathit{body}(\mathit{DMI}) \rightarrow \mathit{SQ}}{S : x = y.m(e); \rightarrow S[x := \mathit{SQ}(\texttt{res}), \$:= \mathit{SQ}(\$), \texttt{exc} := \mathit{SQ}(\texttt{exc})]}$$

$$\frac{S(y) = \texttt{null}}{S : x = y.m(e); \rightarrow S[\$:= S(\$)\langle \texttt{NullPExc} \rangle, \texttt{exc} := \mathit{new}(S(\$), \texttt{NullPExc})]}$$

$$\frac{S[\texttt{par} := S(e), \texttt{res} := \mathit{init}(\mathit{rtyp}(T@m))] : \mathit{body}(T@m) \rightarrow \mathit{SQ}}{S : x = \mathit{super}_T.m(e); \rightarrow S[x := \mathit{SQ}(\texttt{res}), \$:= \mathit{SQ}(\$), \texttt{exc} := \mathit{SQ}(\texttt{exc})]}$$

Well-typed states

$$
\begin{array}{lll}
wts: & Store \rightarrow Bool & \\
 & wts(OS) & \Leftrightarrow \quad \forall InstVar\ IV : \tau(OS(IV)) \leq styp(IV) \\
wt: & State \times ProgPart \rightarrow Bool & \\
 & wt(S, pp) & \Leftrightarrow \quad S(\texttt{this}) \neq \texttt{null} \ \wedge \ wts(S(\$)) \ \wedge \\
 & & \quad \forall VarId\ V : \ V \in vis(pp) \ \Rightarrow \ \tau(S(V)) \leq styp(V, pp) \\
wtp: & State \times ProgPart \rightarrow Bool & \\
 & wtp(S, pp) & \Leftrightarrow \quad wt(S, pp) \ \wedge \ S(\texttt{exc}) = \texttt{null} \\
wtr: & State \times ProgPart \rightarrow Bool & \\
 & wtr(S, pp) & \Leftrightarrow \quad wt(S, pp) \ \wedge \ \tau(S(\texttt{res})) \leq rtyp(pp)
\end{array}
$$

Semantics for program parts

$$sem : State \times ProgPart \times State \rightarrow Bool$$

$$sem(S, c, SQ) \quad \Leftrightarrow \quad wtp(S, c) \ \wedge \ ssem(S, c, SQ)$$

$$sem(S, T@m, SQ) \quad \Leftrightarrow \quad wtp(S, T@m) \ \wedge \ sem(S[\texttt{res} := init(rtyp(T@m))], body(T@m), SQ)$$

$$sem(S, T{:}m, SQ) \quad \Leftrightarrow \quad wtp(S, T{:}m) \ \wedge \ defdm(\tau(S(\texttt{this})), m) \ \wedge \ sem(S, dm(\tau(S(\texttt{this})), m), SQ)$$

Properties of semantics

Lemma (deterministic)

$$sem(S, pp, SQ) \ \wedge \ sem(S, pp, SR) \ \Rightarrow \ SQ = SR$$

Lemma (well-typed poststates)

$$sem(S, pp, SQ) \ \Rightarrow \ wtr(SQ, pp)$$

We also define a fine-grained semantics:

$$rssem \ : \ Nat \times State \times StmtOcc \times State \to Bool$$

Formulas

For every program Π:

- $Th(\Pi)$ denotes the theory defining the types and functions introduced above
- Σ_Π denotes the signature of $TH(\Pi)$
- *assertions* are a first-order formula over $\Sigma_\Pi \cup VarId' \cup \{\$\}$
- *triples* have the form { **P** } *pp* { **Q** } where **P** and **Q** are assertions and *pp* is a program part of Π such that
  - if *pp* is a statement: the program variable are in $vis(pp)$
  - otherwise: preconditions only contain `this`, `par` and postconditions only contain `res`, `exc`

  If a program variable *v* may occur in a pre- or postcondition **R** of *pp*, we say that *v* is *admissible* for **R**.
- *sequents* have the form $\mathcal{A} \mid\!\!\rhd$ **A** where $\mathcal{A}$ is a finite set of method and implementation annotation

Validity

Let F be a formula or term and S be a state.
$F[S]$ denotes the formula or term in which each occurrence of a program variable v is substituted by $S(v')$; similar for \$.
Validity of triples:

$$\{\mathbf{P}\}pp\{\mathbf{Q}\} \equiv_{def} \mathrm{UC}(\forall S, SQ : \mathbf{P}[S] \wedge rsem(N, S, pp, SQ) \Rightarrow \mathbf{Q}[SQ])$$

where N is a logical variable not occurring free in **P** or **Q** and UC denotes the universal closure over all logical variables except N.
Validity of sequents

$$\{\mathbf{P}_1\}m_1\{\mathbf{Q}_1\}, \ldots \models \{\mathbf{P}\}c\{\mathbf{Q}\} \equiv_{def} \forall N : \{\mathbf{P}_1\}\, m_1\{\mathbf{Q}_1\} \wedge \ldots \Rightarrow \{\mathbf{P}\}c\{\mathbf{Q}\}$$

Rules

In the following $\mathcal{A}$ stands for a set of assumptions and **P**, **Q**, **R** are formulas adhering to the restrictions of the corresponding triple

cast-axiom:

$$\mathcal{A} \vdash \left\{ \begin{array}{l} (\tau(e) \leq \mathrm{T} \ \wedge \ \mathbf{P}[e/x]) \quad \vee \\ (\tau(e) \not\leq \mathrm{T} \ \wedge \ \mathbf{P}[\$\langle \mathtt{CastExc} \rangle / \$, \mathit{new}(\$, \mathtt{CastExc})/\mathtt{exc}]) \end{array} \right\} x = (\mathrm{T})e; \ \{ \ \mathbf{P} \ \}$$

field-read-axiom:

$$\mathcal{A} \vdash \left\{ \begin{array}{l} (y \neq \mathtt{null} \ \wedge \ \mathbf{P}[\$(y.a)/x]) \quad \vee \\ (y = \mathtt{null} \ \wedge \ \mathbf{P}[\$\langle \mathtt{NullPExc} \rangle / \$, \mathit{new}(\$, \mathtt{NullPExc})/\mathtt{exc}]) \end{array} \right\} x = y.a; \ \{ \ \mathbf{P} \ \}$$

field-write-axiom:

$$\mathcal{A} \vdash \left\{ \begin{array}{l} (x \neq \mathtt{null} \ \wedge \ \mathbf{P}[\$\langle x.a := e \rangle / \$]) \quad \vee \\ (x = \mathtt{null} \ \wedge \ \mathbf{P}[\$\langle \mathtt{NullPExc} \rangle / \$, \mathit{new}(\$, \mathtt{NullPExc})/\mathtt{exc}]) \end{array} \right\} x.a = e; \ \{ \ \mathbf{P} \ \}$$

Rules (2)

new-axiom:

$$\mathcal{A} \vdash \{\, \mathbf{P}[new(\$, \mathrm{T})/x, \$\langle \mathrm{T} \rangle/\$] \,\}\ x = \texttt{new T()};\ \{\, \mathbf{P} \,\}$$

seq-rule:

$$\frac{\mathcal{A} \vdash \{\, \mathbf{P} \,\}\ s_1\ \{\, (\texttt{exc} \neq \texttt{null} \ \wedge\ \mathbf{Q}) \ \vee\ (\texttt{exc} = \texttt{null} \ \wedge\ \mathbf{R}) \,\} \quad \mathcal{A} \vdash \{\, \mathbf{R} \,\}\ s_2\ \{\, \mathbf{Q} \,\}}{\mathcal{A} \vdash \{\, \mathbf{P} \,\}\ s_1 s_2\ \{\, \mathbf{Q} \,\}}$$

if-rule:

$$\frac{\mathcal{A} \vdash \{\, e = \texttt{true} \ \wedge\ \mathbf{P} \,\}\ s_1\ \{\, \mathbf{Q} \,\} \quad \mathcal{A} \vdash \{\, e = \texttt{false} \ \wedge\ \mathbf{P} \,\}\ s_2\ \{\, \mathbf{Q} \,\}}{\mathcal{A} \vdash \{\, \mathbf{P} \,\}\ \texttt{if}(e)\{s_1\}\texttt{else}\{s_2\}\ \{\, \mathbf{Q} \,\}}$$

while-rule:

$$\frac{\mathcal{A} \vdash \{\, e = \texttt{true} \ \wedge\ \mathbf{I} \,\}\ s\ \{\, \mathbf{I} \,\}}{\mathcal{A} \vdash \{\, \mathbf{I} \,\}\ \texttt{while}(e)\{s\}\ \{\, (\texttt{exc} \neq \texttt{null} \ \vee\ e = \texttt{false}) \ \wedge\ \mathbf{I} \,\}}$$

Rules (3)

catch-rule:

$$\frac{\mathcal{A} \vdash \{\, \mathbf{P} \,\}\ s_0 \left\{ \begin{array}{l} ((\mathtt{exc} = \mathtt{null} \ \vee\ \tau(\mathtt{exc}) \not\leq \mathrm{T}) \ \wedge\ \mathbf{Q}) \\ \vee\ (\mathtt{exc} \neq \mathtt{null} \ \wedge\ \tau(\mathtt{exc}) \leq \mathrm{T} \ \wedge\ \mathbf{R}) \end{array} \right\} \qquad \mathcal{A} \vdash \{\, \mathbf{R}[v/\mathtt{exc}] \,\}\ s_1\ \{\, \mathbf{Q} \,\}}{\mathcal{A} \vdash \{\, \mathbf{P} \,\}\ \mathtt{try}\{s_0\}\mathtt{catch}(\mathrm{T}\ v)\{s_1\}\ \{\, \mathbf{Q} \,\}}$$

block-rule:

$$\frac{\mathcal{A} \vdash \{\, \mathbf{P} \ \wedge\ v = \mathit{init}(\mathrm{T}) \,\}\ s\ \{\, \mathbf{Q} \,\}}{\mathcal{A} \vdash \{\, \mathbf{P} \,\}\ \{\mathrm{T}\ v; s\}\ \{\, \mathbf{Q} \,\}}$$

The triple in the conclusion is only correctly formed if **P** and **Q** do not contain v (v cannot be visible outside the block statement in Java).

Rules (4)

invoc-exc:

$$\vdash \{\ y = \texttt{null}\ \wedge\ \mathbf{Q}[\$\langle \texttt{NullPExc}\rangle/\$, new(\$, \texttt{NullPExc})/\texttt{exc}]\ \}\ x = y.m(e);\ \{\ \mathbf{Q}\ \}$$

invoc-rule:

$$\frac{\mathcal{A} \vdash \{\ \mathbf{P}\ \}\ vm(styp(y', \text{“}x = y.m(e);\text{”}), m)\ \{\ \mathbf{Q}\ \}}{\mathcal{A} \vdash \{\ y \neq \texttt{null}\ \wedge\ \mathbf{P}[y/\texttt{this}, e/\texttt{par}]\ \}\ x = y.m(e);\ \{\ \mathbf{Q}[x/\texttt{res}]\ \}}$$

invoc-var-rule:

$$\frac{\mathcal{A} \vdash \{\ \mathbf{P}\ \}\ x = y.m(e);\ \{\ \mathbf{Q}\ \}}{\mathcal{A} \vdash \{\ \mathbf{P}[w/Z]\ \}\ x = y.m(e);\ \{\ \mathbf{Q}[w/Z]\ \}}$$

where Z is a logical variable and w program variable different from x and $\texttt{exc}$

Rules (5)

call-rule:

$$\frac{\mathcal{A} \vdash \{\ \mathbf{P}\ \}\ T@m\ \{\ \mathbf{Q}\ \}}{\mathcal{A} \vdash \{\ \mathbf{P}[e/par]\ \}\ x = super_T.m(e);\ \{\ \mathbf{Q}[x/res]\ \}}$$

call-var-rule:

$$\frac{\mathcal{A} \vdash \{\ \mathbf{P}\ \}\ x = super_T.m(e);\ \{\ \mathbf{Q}\ \}}{\mathcal{A} \vdash \{\ \mathbf{P}[w/Z]\ \}\ x = super_T.m(e);\ \{\ \mathbf{Q}[w/Z]\ \}}$$

where Z is a logical variable and w a program variable different from x and `exc`.

impl-rule:

$$\frac{\mathcal{A}, \{\ \mathbf{P}\ \}\ T@m\ \{\ \mathbf{Q}\ \} \vdash \{\ \mathbf{P}\ \wedge\ \texttt{res} = init(rtyp(T@m))\ \}\ body(T@m)\ \{\ \mathbf{Q}\ \}}{\mathcal{A} \vdash \{\ \mathbf{P}\ \}\ T@m\ \{\ \mathbf{Q}\ \}}$$

Rules (6)

The proof technique for virtual methods is based on the formalization of the subtype relation and two rules:

$$S \leq T \;\Leftrightarrow\; (\; S = T \vee S \leq T_1 \vee \cdots \vee S \leq T_n \;)$$

where $T_1, \ldots, T_n$ are the subtypes of T

class-rule:

$$\frac{\begin{array}{l} defdm(T, m) \\ \mathcal{A} \vdash \{\, \tau(\texttt{this}) = T \;\wedge\; \mathbf{P} \,\}\; dm(T, m)\; \{\, \mathbf{Q} \,\} \end{array}}{\mathcal{A} \vdash \{\, \tau(\texttt{this}) = T \;\wedge\; \mathbf{P} \,\}\; T{:}m\; \{\, \mathbf{Q} \,\}}$$

subtype-rule:

$$\frac{\begin{array}{l} S \prec T \\ \mathcal{A} \vdash \{\, \mathbf{P} \,\}\; S{:}m\; \{\, \mathbf{Q} \,\} \end{array}}{\mathcal{A} \vdash \{\, \tau(\texttt{this}) \leq S \;\wedge\; \mathbf{P} \,\}\; T{:}m\; \{\, \mathbf{Q} \,\}}$$

Rules (7)

In the following:

- **A** stands for syntactically correct method or implementation annotations
- *Y* and *Z* denote distinct logical variables of the same type
- *pp* denotes program parts

assumpt-axiom:

$$\mathbf{A} \vdash \mathbf{A}$$

false-axiom:

$$\vdash \{ \text{FALSE} \}\ pp\ \{ \text{FALSE} \}$$

assumpt-intro-rule:

$$\frac{\mathcal{A} \vdash \mathbf{A}}{\mathcal{A}, \mathbf{A}_0 \vdash \mathbf{A}}$$

assumpt-elim-rule:

$$\frac{\mathcal{A} \vdash \mathbf{A}_0 \qquad \mathcal{A}, \mathbf{A}_0 \vdash \mathbf{A}}{\mathcal{A} \vdash \mathbf{A}}$$

disjunct-rule:

$$\frac{\mathcal{A} \vdash \{ \mathbf{P1} \}\ pp\ \{ \mathbf{Q1} \} \qquad \mathcal{A} \vdash \{ \mathbf{P2} \}\ pp\ \{ \mathbf{Q2} \}}{\mathcal{A} \vdash \{ \mathbf{P1} \vee \mathbf{P2} \}\ pp\ \{ \mathbf{Q1} \vee \mathbf{Q2} \}}$$

ex-rule:

$$\frac{\mathcal{A} \vdash \{ \mathbf{P} \}\ pp\ \{ \mathbf{Q}[Y/Z] \}}{\mathcal{A} \vdash \{ \exists Z : \mathbf{P} \}\ pp\ \{ \mathbf{Q}[Y/Z] \}}$$

Rules (8)

The substitution in the ex-rule (and also in the all-rule, see below) of Y for Z in **Q** is only used to guarantee that Z does not occur free in **Q**.
In the following:

- v is a program variable, admissible in pre(pp)
- w is program variable, admissible in post(pp)
- t is a Σ-term (no program variables)
- Z logical variable of the same type as t
- **R** is a Σ-formula (no program variables)

strength-rule:

$$\frac{\mathbf{PP} \land \tau(v) \preceq styp(v', pp) \land \texttt{this} \neq \texttt{null} \land \texttt{exc} = \texttt{null} \land wts(\$) \Rightarrow \mathbf{P} \quad \mathcal{A} \vdash \{\ \mathbf{P}\ \}\ pp\ \{\ \mathbf{Q}\ \}}{\mathcal{A} \vdash \{\ \mathbf{PP}\ \}\ pp\ \{\ \mathbf{Q}\ \}}$$

weak-rule:

$$\frac{\mathcal{A} \vdash \{\ \mathbf{P}\ \}\ pp\ \{\ \mathbf{Q}\ \} \quad \mathbf{Q} \land \tau(w) \preceq styp(w', pp) \land \texttt{this} \neq \texttt{null} \land wts(\$) \Rightarrow \mathbf{QQ}}{\mathcal{A} \vdash \{\ \mathbf{P}\ \}\ pp\ \{\ \mathbf{QQ}\ \}}$$

Rules (9)

invar-rule:

$$\frac{\mathcal{A} \vdash \{\ \mathbf{P}\ \}\ pp\ \{\ \mathbf{Q}\ \}}{\mathcal{A} \vdash \{\ \mathbf{P} \wedge \mathbf{R}\ \}\ pp\ \{\ \mathbf{Q} \wedge \mathbf{R}\ \}}$$

subst-rule:

$$\frac{\mathcal{A} \vdash \{\ \mathbf{P}\ \}\ pp\ \{\ \mathbf{Q}\ \}}{\mathcal{A} \vdash \{\ \mathbf{P}[t/Z]\ \}\ pp\ \{\ \mathbf{Q}[t/Z]\ \}}$$

Additional rules:

conjunct-rule:

$$\frac{\mathcal{A} \vdash \{\ \mathbf{P}1\ \}\ pp\ \{\ \mathbf{Q}1\ \} \quad \mathcal{A} \vdash \{\ \mathbf{P}2\ \}\ pp\ \{\ \mathbf{Q}2\ \}}{\mathcal{A} \vdash \{\ \mathbf{P}1 \ \wedge\ \mathbf{P}2\ \}\ pp\ \{\ \mathbf{Q}1 \ \wedge\ \mathbf{Q}2\ \}}$$

all-rule:

$$\frac{\mathcal{A} \vdash \{\ \mathbf{P}[Y/Z]\ \}\ pp\ \{\ \mathbf{Q}\ \}}{\mathcal{A} \vdash \{\ \mathbf{P}[Y/Z]\ \}\ pp\ \{\ \forall Z : \mathbf{Q}\ \}}$$

true-axiom:

$$\vdash \{\ \mathrm{TRUE}\ \}\ pp\ \{\ \mathrm{TRUE}\ \}$$