

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 σ cont sortedPs) ∧
      Ps <~~> sortedPs ) ∧
      (∀x. x ∉ set Ps → 'next x = σ next 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 $\dots, -1I, 0I, 1I, \dots$ 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

```

Statement   = block( Type VarId Statement )
              | cassign( VarId Type Exp )
              | fread( VarId VarId FieldId )
              | fwrite( VarId FieldId VarId )
              | new( VarId CTypeId )
              | seq( Statement Statement )
              | if( Exp Statement Statement )
              | while( Exp Statement )
              | catch( Statement CTypeId VarId Statement )
              | invoc( VarId VarId MethodId Exp )
              | call( VarId CTypeId MethodId Exp )

Exp         = ic( Int ) | bc( Bool ) | nullc | id( VarId )
              | unary( UnOp Exp ) | binary( Exp BinOp Exp )
  
```

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

$$\leq : \text{Type} \times \text{Type} \rightarrow \text{Bool}$$

We assume the types:

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

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

Notations and program parts

In formulas, we write:

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

We assume the types:

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

Helper functions

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

Object store

Formalization of the object store (or heap):

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

Operational semantics

The operational semantics inductively defines the relation $ssem$:

$$ssem : State \times StmtOcc \times State \rightarrow Bool$$

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

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

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

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

States

In Java-KE, we model states as pairs:

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

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 (2)

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

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

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

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

Operational semantics (3)

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

$$S : s_1 s_2 \rightarrow SR$$

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

$$S : s_1 s_2 \rightarrow SQ$$

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

$$S : \text{if}(e)\{s_1\}\text{else}\{s_2\} \rightarrow SQ$$

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

$$S : \text{if}(e)\{s_1\}\text{else}\{s_2\} \rightarrow SQ$$

Operational semantics (4)

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

$$S : \text{while}(e)\{s\} \rightarrow S$$

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

$$S : \text{while}(e)\{s\} \rightarrow SR$$

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

$$S : \text{while}(e)\{s\} \rightarrow SQ$$

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

$$S : \{T v; s\} \rightarrow SQ$$

Operational semantics (5)

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

$$S : \text{try}\{s_0\}\text{catch}(T v)\{s_1\} \rightarrow SQ$$

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

$$S : \text{try}\{s_0\}\text{catch}(T v)\{s_1\} \rightarrow SQ$$

$$\frac{S : s_0 \rightarrow SQ, SQ(\text{exc}) \neq \text{null}, \tau(SQ(\text{exc})) \leq T, SQ[v := SQ(\text{exc}), \text{exc} := \text{null}] : s_1 \rightarrow SR}{S : \text{try}\{s_0\}\text{catch}(T v)\{s_1\} \rightarrow SR}$$

$$S : \text{try}\{s_0\}\text{catch}(T v)\{s_1\} \rightarrow SR$$

Operational semantics (6)

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

$$S : x = y.m(e); \rightarrow S[x := SQ(\text{res}), \$:= SQ(\$), \text{exc} := SQ(\text{exc})]}$$

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

$$S : x = y.m(e); \rightarrow S[\$:= S(\$)(\text{NullPExc}), \text{exc} := \text{new}(S(\$), \text{NullPExc})]}$$

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

$$S : x = \text{super}_T.m(e); \rightarrow S[x := SQ(\text{res}), \$:= SQ(\$), \text{exc} := SQ(\text{exc})]}$$

Well-typed states

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

Semantics for program parts

$$\begin{aligned}
 sem &: State \times ProgPart \times State \rightarrow Bool \\
 sem(S, c, SQ) &\Leftrightarrow wtp(S, c) \wedge ssem(S, c, SQ) \\
 sem(S, T@m, SQ) &\Leftrightarrow wtp(S, T@m) \wedge \\
 &\quad sem(S[\text{res} := \text{init}(rtyp(T@m))], \text{body}(T@m), SQ) \\
 sem(S, T:m, SQ) &\Leftrightarrow wtp(S, T:m) \wedge \text{defdm}(\tau(S(\text{this})), m) \wedge \\
 &\quad sem(S, \text{dm}(\tau(S(\text{this})), m), SQ)
 \end{aligned}$$

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:

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

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 $\{ \mathbf{P} \} pp \{ \mathbf{Q} \}$ where \mathbf{P} and \mathbf{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 \mathbf{R} of pp , we say that v is *admissible* for \mathbf{R} .

- *sequents* have the form $\mathcal{A} \vdash \mathbf{A}$ 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:

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

where N is a logical variable not occurring free in \mathbf{P} or \mathbf{Q} and UC denotes the universal closure over all logical variables except N .

Validity of sequents

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

Rules (2)

new-axiom:

$$\vdash \{ \mathbf{P}[new(\$, T)/x, \$(T)/\$] \} x = new \ T(); \{ \mathbf{P} \}$$

seq-rule:

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

if-rule:

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

while-rule:

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

Rules

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

cast-axiom:

$$\vdash \left\{ \begin{array}{l} (\tau(e) \leq T \wedge \mathbf{P}[e/x]) \vee \\ (\tau(e) \not\leq T \wedge \mathbf{P}[\$(\text{CastExc})/\$, new(\$, \text{CastExc})/\text{exc}]) \end{array} \right\} x = (T)e; \{ \mathbf{P} \}$$

field-read-axiom:

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

field-write-axiom:

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

Rules (3)

catch-rule:

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

block-rule:

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

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

Rules (4)

invoc-exc:

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

invoc-rule:

$$\mathcal{A} \vdash \{ \mathbf{P} \} \text{vm}(\text{styp}(y', "x = y.m(e);"), m) \{ \mathbf{Q} \}$$

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

invoc-var-rule:

$$\mathcal{A} \vdash \{ \mathbf{P} \} x = y.m(e); \{ \mathbf{Q} \}$$

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

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

Rules (5)

call-rule:

$$\mathcal{A} \vdash \{ \mathbf{P} \} T@m \{ \mathbf{Q} \}$$

$$\mathcal{A} \vdash \{ \mathbf{P}[e/\text{par}] \} x = \text{super}_T.m(e); \{ \mathbf{Q}[x/\text{res}] \}$$

call-var-rule:

$$\mathcal{A} \vdash \{ \mathbf{P} \} x = \text{super}_T.m(e); \{ \mathbf{Q} \}$$

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

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

impl-rule:

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

$$\mathcal{A} \vdash \{ \mathbf{P} \} T@m \{ \mathbf{Q} \}$$

Rules (6)

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

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

where T_1, \dots, T_n are the subtypes of T

class-rule:

$$\text{defdm}(T, m)$$

$$\mathcal{A} \vdash \{ \tau(\text{this}) = T \wedge \mathbf{P} \} \text{dm}(T, m) \{ \mathbf{Q} \}$$

$$\mathcal{A} \vdash \{ \tau(\text{this}) = T \wedge \mathbf{P} \} T:m \{ \mathbf{Q} \}$$

subtype-rule:

$$S < T$$

$$\mathcal{A} \vdash \{ \mathbf{P} \} S:m \{ \mathbf{Q} \}$$

$$\mathcal{A} \vdash \{ \tau(\text{this}) \leq S \wedge \mathbf{P} \} T:m \{ \mathbf{Q} \}$$

Rules (7)

In the following:

- \mathbf{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:

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

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

assumpt-elim-rule:

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

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

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

disjunct-rule:

$$\mathcal{A} \vdash \{ \mathbf{P}_1 \} pp \{ \mathbf{Q}_1 \}$$

$$\mathcal{A} \vdash \{ \mathbf{P}_2 \} pp \{ \mathbf{Q}_2 \}$$

$$\mathcal{A} \vdash \{ \mathbf{P}_1 \vee \mathbf{P}_2 \} pp \{ \mathbf{Q}_1 \vee \mathbf{Q}_2 \}$$

ex-rule:

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

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

Rules (8)

The substitution in the ex-rule (and also in the all-rule, see below) of Y for Z in \mathbf{Q} is only used to guarantee that Z does not occur free in \mathbf{Q} .

In the following:

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

strength-rule:

$$\frac{\mathbf{PP} \wedge \tau(v) \leq \text{styp}(v', pp) \wedge \text{this} \neq \text{null} \wedge \text{exc} = \text{null} \wedge \text{wts}(\$) \Rightarrow \mathbf{P}}{\mathcal{A} \vdash \{ \mathbf{P} \} pp \{ \mathbf{Q} \}}$$

$$\mathcal{A} \vdash \{ \mathbf{PP} \} pp \{ \mathbf{Q} \}$$

weak-rule:

$$\mathcal{A} \vdash \{ \mathbf{P} \} pp \{ \mathbf{Q} \}$$

$$\mathbf{Q} \wedge \tau(w) \leq \text{styp}(w', pp) \wedge \text{this} \neq \text{null} \wedge \text{wts}(\$) \Rightarrow \mathbf{QQ}$$

$$\mathcal{A} \vdash \{ \mathbf{P} \} pp \{ \mathbf{QQ} \}$$

Rules (9)

invar-rule:

$$\mathcal{A} \vdash \{ \mathbf{P} \} pp \{ \mathbf{Q} \}$$

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

subst-rule:

$$\mathcal{A} \vdash \{ \mathbf{P} \} pp \{ \mathbf{Q} \}$$

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

Additional rules:

conjunct-rule:

$$\mathcal{A} \vdash \{ \mathbf{P1} \} pp \{ \mathbf{Q1} \}$$

$$\mathcal{A} \vdash \{ \mathbf{P2} \} pp \{ \mathbf{Q2} \}$$

$$\mathcal{A} \vdash \{ \mathbf{P1} \wedge \mathbf{P2} \} pp \{ \mathbf{Q1} \wedge \mathbf{Q2} \}$$

all-rule:

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

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

true-axiom:

$$\vdash \{ \text{TRUE} \} pp \{ \text{TRUE} \}$$