Soundness and Relative Completeness of a Programming Logic for a Sequential Java Subset

(Preliminary version)

Arnd Poetzsch-Heffter, Jean-Marie Gaillourdet, Nicole Rauch
Technische Universität Kaiserslautern,
67653 Kaiserslautern, Germany
[poetzsch,jmg,rauch]@informatik.uni-kl.de

January 20, 2005

Abstract

This report presents an operational semantics and a programming logic for a sequential Java subset. The subset covers all essential sequential language concepts, like classes, interfaces, subtyping, dynamic binding, and a simple form of exception handling. The logical framework supports user-defined specification parts and abstraction. The report concentrates on the soundness and relative completeness proof of the programming logic.

1 Introduction

In this report, we present a programming logic for a sequential Java **k**ernel with a simple **e**xception mechanism. We call the language *Java-KE*. Java-KE is a typical object-oriented kernel language. Its abstract syntax and operational semantics are described in Section 2. Section 3 contains an equivalent operational semantics that simplifies the proofs in the subsequent sections.

The programming logic for Java-KE is based on Hoare triples. It was motivated and developed for the Jive system ([7]), an interactive verification environment for Java programs. The pre- and postconditions of the Hoare triples are sorted first-order formulas. The object store (or heap) is handled as a global variable with suitable access functions. The programming logic is presented in Section 4. The main contribution of this report is to make the proofs of soundness (Section 5) and relative completeness (Section 6) of the logic publicly available.

In the following, we assume that the reader is familiar with Java, with techniques for specifying operational semantics, with sorted first-order logic, and with Hoare logic.

Related Work. [9] [1] [12] [6]

2 The Language Java-KE

This section contains the specification of the Java sublanguage Java-KE. We present the abstract syntax, static semantics, and operational semantics.

2.1 Syntax of Java-KE

In this report, we do not consider packages or compilation units, although the logic was developed for a Java subset with packages. Thus, a Java-KE program is simply a list of type declarations where a type declaration is an interface or class declaration.

Figure 1 shows the abstract syntax of Java-KE. As we assume that the reader is familiar with Java, we omit a detailed explanation and concentrate on the restrictions and technical issues. We assume the following sorts: *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); furthermore, the sort *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. The operators in *UnOp* and *BinOp* do not cause side-effects and do not raise exceptions; in particular, *BinOp* does not contain the division and modulo operators (see Section 7 for language extensions).

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

Figure 1: Abstract syntax of Java-KE

A class declaration consists of the declared name of the class, the superclass name, the list of interfaces it implements, and the body. In case of the special class `Object` that has no superclass, the second component can be arbitrary and has no meaning. Java-KE only supports methods with one parameter and with a return value. The parameter is implicitly named `par`. The only reason for this simplification is to avoid unnecessary overhead in the presentation. The extension of the specifications, the logic, and the proofs to methods without parameters or return values and to methods with several parameters is straightforward. A type in Java-KE is a predefined

type (for convenience, we write `boolean` for *booleanT* and `int` for *intT*), the special type `Null` for typing the null-reference (we write `Null` instead of *nullT*), a class type, or an interface type. A statement is:

- a block with one variable declaration,
- an assignment, possibly with a cast, i.e. of the form "$x = e$;" or "$x = (T)e$;" ,
- a field read of the form "$x = y.a$;" ,
- a field write of the form "$x.a = y$;" ,
- a creation expression with the default constructor of the form "$x = new\ T()$;" ,
- a sequential, conditional, or loop statement,
- a catch statement of the form "`try { `s_1` } catch(` *ExcType* z `) { `s_2` }`" where *ExcType* denotes either the type `CastExc` for cast exceptions or `NullPExc` for null pointer exceptions (see below),
- a dynamically bound method invocation, or
- a statically bound method invocation; in Java-KE statically bound method invocations[1] represent calls of overridden methods in superclasses of the form "$x =$ `super`$.m(e)$;"; the second component in the abstract syntax denotes the superclass T in which the called method is declared. Note that T can be statically determined. To make T explicit, we will write "$x =$ `super`$_T.m(e)$;" in the following.

Expressions in Java-KE are either constants, variables, or nested applications of side-effect free operators.

In the following, we will denote Java-KE programs either in the abstract syntax described above or in the concrete syntax of Java, depending on what seems more readable to us.

Context Conditions. In addition to what is required for every Java program, Java-KE programs have to satisfy the following context conditions:

1. Declared local variables must not be named `exc` or `res`; the special meaning of these variables is explained below. `res` is a local variable that is implicitly declared in every method (like the standard parameter identifiers `this` and `par`).
2. Field identifiers are unique, i.e. fields in different classes have to be named differently (e.g. by preceding them with the name of the enclosing class).
3. Java-KE does not support overloading of methods.

We assume that every Java-KE program contains a class `Object` with at least one method and one field. (Thus, every program has at least one method and each object has at least one field. The reason for this is to guarantee that the sorts defined in the following are non-empty.) In addition, every Java-KE program contains classes `CastExc` and `NullPExc` to create class cast and null pointer exceptions.

As Java-KE neither supports static methods nor strings, we have to use a different startup convention. Each program has to contain a class `Main` with a method `main` taking an integer parameter `par` as argument and yielding an integer `res` as result. Program execution starts by creating an object of class `Main` and invoking method `main` on that object with the input as argument. That is, a program execution corresponds to the execution of the following block statement:

[1] In Java, calls of static methods and private methods are also statically bound.

```
{
   Main mvr;
   mvr = new Main();
   res = mvr.main( par );
}
```

We assume that this statement is implicitly part of each program and call it the *startup statement*.

2.2 Static Semantics of Java-KE

The static semantics specifies the set of program states and further notions needed for the dynamic semantics. For Java-KE, the static semantics includes the sorts and signatures of the operations modelling the object store. To keep the explanations simpler, we assume that a Java-KE program Π is given, and interpret the definitions with respect to Π. In particular, the identifier sorts *FieldId*, *MethodId*, *CTypeId*, *ITypeId*, and *VarId* correspond to the identifiers occurring in Π.

The subtype relation on the types of Π is defined as in Java; we denote it by the binary predicate $\preceq$:

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

We assume that a complete axiomatization of $\preceq$ is given in the logical framework (see below). Note that this axiomatization is finite, because Π has only a finite number of types. By $\prec$, we denote the proper subtype relation.

The sort of object identifiers is *ObjId*; we assume that *ObjId* is linearly ordered and has a successor function *next* (e.g. the natural numbers could be used as a represenation of *ObjId*). An object is formally represented by a pair consisting of identifier and an object identifier. The main reason for using such pairs is to capture the type of an object in its representation. Based on this modeling decision, the set of values of Java-KE can be defined by the following datatype:

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

A value is a boolean, an integer, the null reference, or a reference to an object of a certain class with a certain identifier. Instead of b(TRUE) and b(FALSE), we write `true` and `false`, instead of $i(-1I)$, $i(0I)$, $i(1I)$, we write -1, 0, 1. Instead of the constructor *null*, we use as well the constant `null` to stress that this is the same value as the null-reference in Java. According to its class declaration, every object has a fixed number of typed instance variables. The set of all instance variables of all objects is captured by the sort *InstVar*. To refer precisely to methods and statements in program Π, we introduce the following sorts:

- The set of all method declarations in Π is captured by the sort *DeclMethId*; the elements of *DeclMethId* will be denoted by strings of the form *cid*@*mid* where *cid* is a class identifier in *CTypeId* and *mid* $\in$ *MethodId* is an identifier of a method declared in *cid*.
- Analogously, the set of so-called *virtual methods* is captured by the sort *VirtMethId* with elements of the form *tid*:*mid* such that the following holds: *tid*:*mid* $\in$ *VirtMethId* iff *tid* is an interface containing a method signature for *mid*, or *tid* is a class containing a method declaration for *mid*, or type *tid* inherits a method with identifier *mid*.
- The set of all statement occurrences in Π as well as the startup statement with its substatements is captured by sort *StmtOcc*.
- The disjoint union of *DeclMethId*, *VirtMethId*, and *StmtOcc* is captured by sort *ProgPart*. Note that there are no method declaration occurrences. Methods are only referenced by their identifier, not by their position. This is because the order of the methods in the source code is irrelevant to the verification process.

The elements of sort *ProgPart* are those parts of a program that can be specified by a pre- and a postcondition, that is, they can appear as second component of a Hoare triple (see Section 4). In particular, a Hoare triple can express a property of a declared or virtual method. To formulate some restrictions on the use of program variables in Hoare triples, we assume a function

$$vis: \ ProgPart \to 2^{VarId}$$

that yields for each program part the set of visible variable identifiers: For statement occurrences c, $vis(c)$ includes `res`, `exc`, `this`, and `par` as well as the local variables visible at c. For declared or virtual method identifiers m, $vis(m) = \{\texttt{this}, \texttt{par}, \texttt{exc}\}$.[2]
Furthermore, we need the following functions:

τ	: *Value*	$\to$	*Type*
init	: *Type*	$\to$	*Value*
styp	: *InstVar*	$\to$	*Type*
styp	: *VarId* $\times$ *ProgPart*	$\to$	*Type*
rtyp	: *ProgPart*	$\to$	*Type*
defdm	: *Type* $\times$ *MethodId*	$\to$	*Bool*
dm	: *Type* $\times$ *MethodId*	$\to$	*DeclMethId*
defvm	: *Type* $\times$ *MethodId*	$\to$	*Bool*
vm	: *Type* $\times$ *MethodId*	$\to$	*VirtMethId*
body	: *DeclMethId*	$\to$	*StmtOcc*
$\_.\_$	: *Value* $\times$ *FieldId*	$\to$	*InstVar*

These functions are defined as follows:

- τ yields the type of a value; in particular, $\tau(\texttt{null}) = \texttt{Null}$.
- *init* yields for each type an initial value: `true` for type `boolean`, 0 for type `int`, and `null` for the reference types.
- applied to an instance variable *iv* of the implicitly given program Π, *styp* yields *iv*'s static type; applied to a variable identifier *vid* visible at a program part *pp*, *styp* yields the static type of *vid* in *pp*; for the variable `res`, it yields the result type of the enclosing method; for the exception variable `exc`, we define $styp(\texttt{exc}, pp) = ct(\texttt{Object})$; [3] if *vid* is not visible at *pp*, $styp(vid, pp)$ is arbitrary.
- If applied to a statement occurrence of Π, *rtyp* yields the result type of the enclosing method (i.e. the type of `res`), otherwise the return type of the declared or virtual method.
- The predicates *defdm* and *defvm* check whether there exists a declared method or virtual method in Π for the given type T and method identifier m:
  - $defdm(T, m)$ yields true if a method implementation with identifer m is declared in or inherited by class T. If so, $dm(T, m)$ yields the corresponding element $S@m$ of sort *DeclMethId*: $S = T$ if m is declared in T, otherwise S is the closest superclass with a declaration for m.
  - $defvm(T, m)$ yields true if a method signature or implementation with identifer m is declared in or inherited by type T. If so, $vm(T, m)$ yields $T : m$ of sort *VirtMethId*.

  If $defdm(T, m)$ or $defvm(T, m)$ yield false, the results of $dm(T, m)$ and $vm(T, m)$ resp. are arbitrary.[4]

[2] For the visibility of program variables in the pre- and postconditions of the Hoare logic, refer to Sec. 4.1, p. 17.

[3] For the notation see the abstract syntax of Java-KE in Fig. 1, p. 3.

[4] As the context conditions guarantee that class `Object` has a method, both *DeclMethId* and *VirtMethId* are non-empty.

- *body* yields the body of a declared method in Π, that is, the statement describing the implementation of the method.
- The field selection operation $x.f$ selects the instance variable f of object x; if x is not an object or has no field f, the result is arbitrary.

One of the most interesting aspects of the semantics of object-oriented languages is the formalization of the object store or heap. We use an abstract datatype with sort *Store* to specify states of the object store. The state of an object store captures the state of the instance variables and the information whether an object is allocated or not. The datatype has the following five operations:

- $\_(\_) : \mathit{Store} \times \mathit{InstVar} \to \mathit{Value}$ 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 $OS\langle \mathit{TID}\rangle$ yields the state of the object store after allocating a new object of type $ct(\mathit{TID})$.
- $\mathit{new} : \mathit{Store} \times \mathit{CTypeId} \to \mathit{Value}$ where $\mathit{new}(OS, \mathit{TID})$ yields the object of type $ct(\mathit{TID})$ 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.

As the operational semantics and logic use the same datatype for the object store, the details of its specification are irrelevant for the proofs of this report. That is, this work is parametric w.r.t. the object store datatype which makes it applicable to different object store realizations. A possible axiomatization of the datatype and its operations is given in [11], Sect. 3.1.2.

2.3 Operational Semantics

The operational semantics specifies how the program state evolves during computations. In JavaKE, we model states as pairs:

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

The first component maps variable identifiers to their current value, the second component yields the current state of the object store where \$ is considered a global variable of sort *Store*. Application of a state S to a *VarId* v is denoted by $S(v)$, to the store variable \$ by $S(\$)$. Update of S at v or \$ is written $S[v := E]$ or $S[\$:= E]$ respectively. States are usually denoted by S, SP, SQ, SR. We extend states canonically to expressions: If e is a program expression of sort *Exp*, we write $S(e)$ for the evaluation of e in S.
The rules of the operational semantics inductively define the relation:

$$\mathit{ssem} \quad : \quad \mathit{State} \times \mathit{StmtOcc} \times \mathit{State} \to \mathit{Bool}$$

where *ssem* stands for statement semantics. For states SP, SQ, and a statement occurrence c in Π, $\mathit{ssem}(SP, c, SQ)$ expresses the fact that execution of c in SP terminates with a poststate SQ. If execution of c has completed normally, we have $SQ(\texttt{exc})=\texttt{null}$. $SQ(\texttt{exc})\neq \texttt{null}$ indicates abrupt completion: Depending on the object held by $SQ(\texttt{exc})$ either a class cast exception or a null pointer exception has occurred. The relation *ssem* does not presume type safety. Essentially, there are three reasons why we start with a semantics for arbitrary states and consider well-typedness and type safety as a second step:

- Whether a state is well-typed depends on its statement context, because the statement context determines the types of the local variables. Thus, quantification over well-typed states has to take statement contexts into account and becomes technically more complex.

- Defining semantics in terms of untyped states allows to prove type safety as a property leading to a clearer separation between the definition and proved properties.
- There are special properties for prestates that have to be handled in a second step anyway.

Within the semantical rules, we use concrete syntax for statements and the convention to write $SP : c \to SQ$ instead of $ssem(SP, c, SQ)$. We start with the rules for the classical statements (assignment-, if-, while-statement) and give the typical object-oriented statements at the end.

In Java-KE, casts are only allowed at assigments to local variables. Depending on the outcome of the cast, the assignment completes normally or throws an exception:

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

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

Reading and writing locations has to take the possibility of dereferencing `null` into account. If the dereferenced value is `null`, a `NullPExc`-object is created and assigned to `exc`.

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

Note that this rule does not set `exc` to `null` in the poststate. Thus, if $S(\texttt{exc}) \neq \texttt{null}$ in the prestate, we would observe abrupt termination in the poststate, although the exception was not caused by the assignment. As we want to exclude this misleading behavior, we are only interested in prestates S with $S(\texttt{exc}) = \texttt{null}$. In the following, we will treat this restriction on prestates similar to typing conditions (see below).

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

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

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

In Java-KE object creation always terminates normally:

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

The following semantic rules describe the classical imperative statements and how they treat exceptions:

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

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

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

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

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

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

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

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

Similar to the classical statement, the semantics of Java-KE's try-catch-statement is formulated by testing the special variable `exc`:

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

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

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

The most interesting rules are those for method invocation. As we cannot assume type safety in the operational semantics, we formulate the corresponding constraint as a premise of the first rule:

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

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

While in the rule above the called method depends on the type $\tau(S(y))$ of the current target object, "super"-calls to methods in superclasses are statically bound:

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

2.4 Execution on Well-Typed States

In this subsection, we investigate properties of Java-KE and its semantics. In particular, we discuss type safety. Central are the notions of well-typed stores and well-typed states. We use the following predicates to express well-typedness where *wts* stands for well-typed stores, *wt* for well-typed states, *wtp* for well-typed and -formed prestates, and *wtr* for well-typed poststates of methods:

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

To be a well-formed prestate S of a program part, it is required that the special variable `exc` is null. The predicate *wtr* is helpful to abbreviate well-typedness for poststates of methods (recall that for a method m: $\texttt{res} \notin vis(m)$). For statement occurrences c, it holds that $wtr(S, c) \Leftrightarrow wt(S, c)$, because $\texttt{res} \in vis(c)$. Based on *ssem* and the well-typedness predicates, we define a semantic relation that applies to statements and method abstractions:

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

The following two lemmas state properties of Java-KE that we will need in the main proofs of subsequent sections.

Lemma 1 (deterministic)**.** *Java-KE is deterministic, that is, sem is a partial function from states and program parts to states. More precisely:*

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

Proof by induction on the depth of the semantic derivation tree. (The first two arguments determine the tree structure.)

Lemma 2 (well-typed poststates)**.** *Terminating Java-KE executions that started in a well-typed state always result in a well-typed state. More precisely:*

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

Type soundness can be shown as usual. We haven't worked out the proof in this report. However, in an Isabelle HOL formalization of the language and its semantics, the type soundness is formally proven.

3 Refined Operational Semantics

To enable the direct embedding of Hoare sequents into the semantics, we define a slightly richer semantic relation *rssem*:

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

where the first parameter captures the maximal depth of nested method calls that is allowed during execution of the statement. That is, for states SP, SQ, and a statement occurrence c in a program Π, $rssem(N, SP, c, SQ)$ expresses the fact that execution of c in SP does not lead to more than N nested calls and terminates with a poststate SQ. We write $S:\ c\ \to_N\ SQ$ instead of $rssem(N, S, c, SQ)$. As it will be explained in Sect. 4, the additional parameter N captures the inductive argument that is hidden behind the elegant notation of sequents in Hoare logic.

The rules defining *rssem* are the same as those for *ssem* given in the last section, except for the additional parameter N. If the rule does not describe the semantics of a call, we simply replace *ssem* by *rssem* and add a free variable N. Here is a typical example:

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

Only in the rules that describe the execution of method calls, the parameter N is affected:

$$\frac{\begin{array}{l}S(y) \neq \texttt{null},\ \tau(S(y)) \preceq styp(y, \text{“}x = y.m(e);\text{”}),\ DMI = dm(\tau(S(y)), m),\\ S[\texttt{this} := S(y), \texttt{par} := S(e), \texttt{res} := init(rtyp(DMI))]:\ body(DMI)\ \to_N\ SQ\end{array}}{S:\ x = y.m(e);\ \to_{N+1}\ S[x := SQ(\texttt{res}),\ \$:= SQ(\$),\ \texttt{exc} := SQ(\texttt{exc})]}$$

$$\frac{S[\texttt{par} := S(e), \texttt{res} := init(rtyp(dm(T, m)))]:\ body(dm(T, m))\ \to_N\ SQ}{S:\ x = super_T.m(e);\ \to_{N+1}\ S[x := SQ(\texttt{res}),\ \$:= SQ(\$),\ \texttt{exc} := SQ(\texttt{exc})]}$$

Similar to *sem*, we define a derived semantic predicate *rsem* for statements and methods:

$$\begin{array}{lll}
rsem \;:\; Nat \times State \times ProgPart \times State \to Bool & & \\
rsem(N, S, c, SQ) & \Leftrightarrow & wtp(S, c)\ \wedge\ rssem(N, S, c, SQ)\\
rsem(0, S, T@m, SQ) & \Leftrightarrow & false\\
rsem(N+1, S, T@m, SQ) & \Leftrightarrow & wtp(S, T@m)\ \wedge\\
 & & rsem(N, S[\texttt{res} := init(rtyp(T@m))], body(T@m), SQ)\\
rsem(N, S, T{:}m, SQ) & \Leftrightarrow & wtp(S, T{:}m)\ \wedge\ defdm(\tau(S(\texttt{this})), m)\ \wedge\\
 & & rsem(N, S, dm(\tau(S(\texttt{this})), m), SQ)
\end{array}$$

As we consider *ssem* as the defining semantics of Java-KE, we need some properties relating *rssem* to *ssem*. Otherwise, we would not be able use *rssem* in soundness proofs. The properties are stated as lemmas. For the proofs of the lemmas, we translate the rules of the **s**tructured **o**perational **s**ematics (SOS-rules for short) for *ssem* and *rssem* into recursive predicate definitions. We sketch this here for *rssem*:

$$rssem(N, SP, c, SR) \Leftrightarrow_{def} \bigvee_{R \in SOS\text{-}rules} ((c \text{ matches } stmtpattern(R)) \wedge antecedents(R) \wedge poststate(R))$$

where $stmtpattern(R)$ is the statement pattern occurring in the succedent of R, $antecedents(R)$ denotes the antecedents of the rule where free occurrences of logical variables are existentially bound, and $poststate(R)$ denotes the definition of the poststate SR. If the definition of the poststate SR is trivial, it is directly substituted in $stmtpattern(R)$. The prestate SP is substituted

in $stmtpattern(R)$ in any case. We demonstrate this here by two disjuncts, namely the disjunct for one of the sequential statement composition rules and that for the central method invocation rule:

$$\begin{array}{l} rssem(N, SP, c, SR) \ \Leftrightarrow_{def} \\ c \text{ matches } stmtpattern(s_1 \ s_2) \ \wedge \\ rssem(N, \ SP, \ s_1, SR) \ \wedge \ SR(\texttt{exc}) \neq \texttt{null} \end{array}$$

$$\begin{array}{l} rssem(N, SP, c, SR) \ \Leftrightarrow_{def} \\ c \text{ matches } stmtpattern(x = y.m(e);) \ \wedge \ N > 0 \ \wedge \\ \exists \ SQ, \ DMI : SP(y) \neq \texttt{null} \ \wedge \ \tau(SP(y)) \preceq styp(y, c) \ \wedge \ DMI = dm(\tau(SP(y)), m) \\ \quad \wedge \ rssem(N-1, SP[\texttt{this} := SP(y), \texttt{par} := SP(e), \texttt{res} := init(rtyp(DMI))], body(DMI), \ SQ) \\ \quad \wedge \ SR = SP[x := SQ(\texttt{res}), \$:= SQ(\$), \texttt{exc} := SQ(\texttt{exc})] \end{array}$$

The proofs of the lemmas are based on notions from fixed point theory for first-order logic. In particular, we use the functionals ϕ_{ssem} and ϕ_{rssem} corresponding to the above recursive predicate definitions. Applying the functionals j times to the predicate *FALSE* that is false everywhere is denoted by $ssem_j$ and $rssem_j$, that is in particular:

$$rssem_j(N, S, c, SQ) \quad \Leftrightarrow \quad \phi^j_{rssem}(\ FALSE(N, S, c, SQ)\ )$$

Lemma 3 (monotone)**.**

$$rssem(N, S, c, SQ) \Rightarrow rssem(N+1, S, c, SQ)$$

Proof:
By induction, we show for all $j \geq 0$ that the following implication holds for arbitrary *N,S,c,SQ*:

$$rssem_j(N, S, c, SQ) \Rightarrow rssem_j(N+1, S, c, SQ)$$

$j = 0$: Both sides of the implication are false independent of *S, SQ, c, N*.
$j \rightarrow j+1$: Induction hypothesis: $rssem_j(N, S, c, SQ) \Rightarrow rssem_j(N+1, S, c, SQ)$. Let M be arbitrary:

$rssem_{j+1}(M, S, c, SQ)$
$\Rightarrow$ [[definition of $rssem_{j+1}$]]
One of the disjuncts holds; let's denote this disjunct by D;
D has only positive ocurrences of "$M > 0$", if any, and of "$rssem_j(M', S', c', sq')$"
where M' is M or $M-1$ (see disjunct example above).
$\Rightarrow$ [[only positive occurrences; $M > 0 \Rightarrow M + 1 > 0$; induction hypothesis]]
The formula resulting from replacing M by $M+1$ in D holds as well.
$\Rightarrow$ [[definition of $rssem_{j+1}$]]
$rssem_{j+1}(M+1, S, c, SQ)$

QED.

Lemma 4 (ssem-rssem)**.**

$$ssem(S, c, SQ) \Leftrightarrow \exists N : rssem(N, S, c, SQ)$$

proof:
by induction, we show for all $j \geq 0$ that the following equivalence holds for arbitrary *N,S,c,SQ*:

$$ssem_j(S, c, SQ) \leftrightarrow \exists \ N : \ rssem_j(N, S, c, SQ)$$

$j = 0$: both sides of the equivalence are false independent of *S,c,SQ*.
$j \rightarrow j+1$: Induction hypothesis: $ssem_j(S, c, SQ) \leftrightarrow \exists \ N : rssem_j(N, S, c, SQ)$.

$ssem_{j+1}(S, c, SQ)$
$\Leftrightarrow$ [[definition of *ssem*]]
one of the disjuncts holds.
$\Leftrightarrow$ [[induction hypothesis]]
the formula obtained by replacing all occurrences of $ssem_j(S', c', SQ')$ in the disjunct by $\exists\ N : rssem_j(N, S', c', SQ')$ holds as well.
$\Leftrightarrow$ [[only positive occurrences; lemma *monotone*; possibly renaming]]
The formula obtained by placing the existential quantifier for N at outermost position and conjoining $N > 0$, if appropriate, holds.
$\Leftrightarrow$ [[definition of *rssem*]]
$\exists\ N : rssem_{j+1}(N, S, c, SQ)$
QED.

Lemma 5 (monoton-rsem)**.** $rsem(N, S, pp, SQ)\ \Rightarrow\ rsem(N + 1, S, pp, SQ)$

Proof:
Case distinction over the form of *pp*:

Case a: *pp* is a statement.
Follows from definition of *rsem* and lemma *monoton*.

Case b: *pp* is a method implementation $T@m$.
In case $N = 0$, the premise is false. Otherwise let $M + 1 = N > 0$:

$rsem(N, S, T@m, SQ)$
$\Leftrightarrow$ [[definition of M]]
$rsem(M + 1, S, T@m, SQ)$
$\Leftrightarrow$ [[definition of rsem]]
$wtp(S, T@m)\ \wedge\ rsem(M, S[\texttt{res} := init(rtyp(T@m))], body(T@m), SQ)$
$\Rightarrow$ [[case a]]
$wtp(S, T@m)\ \wedge\ rsem(M + 1, S[\texttt{res} := init(rtyp(T@m))], body(T@m), SQ)$
$\Leftrightarrow$
$rsem(M + 2, S, T@m, SQ)$
$\Leftrightarrow$
$rsem(N + 1, S, T@m, SQ)$

Case c: *pp* is a virtual method *T:m*.

$rsem(N, S, T{:}m, SQ)$
$\Leftrightarrow$ [[definition of rsem]]
$wtp(S, T{:}m)\ \wedge\ defdm(\tau(S(\texttt{this})), m)\ \wedge\ rsem(N, S, dm(\tau(S(this)), m), SQ)$
$\Rightarrow$ [[case b]]
$wtp(S, T{:}m)\ \wedge\ defdm(\tau(S(\texttt{this})), m)\ \wedge\ rsem(N + 1, S, dm(\tau(S(this)), m), SQ)$
$\Leftrightarrow$
$rsem(N + 1, S, T{:}m, SQ)$

QED.

Lemma 6 (sem-rsem)**.** $sem(S, pp, SQ) \Leftrightarrow \exists N : rsem(N, S, pp, SQ)$

Proof:
Case distinction over the form of *pp*:

Case a: *pp* is a statement.
Follows from definition of *sem*, *rsem*, and lemma *ssem-rssem*.

Case b: *pp* is a method implementation $T@m$.

$$
\begin{array}{ll}
 & sem(S, T@m, SQ) \\
\Leftrightarrow & \text{[[definition of sem]]} \\
 & wtp(S,\, T@m) \ \wedge \ sem(S[\texttt{res} := init(rtyp(\,T@m))], body(\,T@m), SQ) \\
\Rightarrow & \text{[[case a]]} \\
 & \exists N: \ wtp(S,\, T@m) \ \wedge \ rsem(N, S[\texttt{res} := init(rtyp(\,T@m))], body(\,T@m), SQ) \\
\Leftrightarrow & \text{[[definition of rsem]]} \\
 & \exists N : rsem(N+1, S, T@m, SQ) \\
\Leftrightarrow & \\
 & \exists N : N > 0 \ \wedge \ rsem(N, S, T@m, SQ) \\
\Leftrightarrow & \text{[[definition of rsem]]} \\
 & \exists N : rsem(N, S, T@m, SQ)
\end{array}
$$

Case c: *pp* is a virtual method *T:m*.

$$
\begin{array}{ll}
 & sem(S,\, T{:}m, SQ) \\
\Leftrightarrow & \text{[[definition of sem]]} \\
 & wtp(S,\, T{:}m) \ \wedge \ defdm(\tau(S(\texttt{this})), m) \\
 & \quad \wedge \ sem(S, dm(\tau(S(\texttt{this})), m), SQ) \\
\Leftrightarrow & \text{[[case b]]} \\
 & \exists N: \ wtp(S,\, T{:}m) \ \wedge \ defdm(\tau(S(\texttt{this})), m) \\
 & \quad \wedge \ rsem(N, S, dm(\tau(S(\texttt{this})), m), SQ) \\
\Leftrightarrow & \text{[[definition of rsem]]} \\
 & \exists N : rsem(N, S,\, T{:}m, SQ)
\end{array}
$$

QED: sem-rsem.

Lemma 7 (well-typed-rres)**.**

$$rsem(N, S, pp, SQ) \Rightarrow wtr(SQ, pp)$$

Proof:
The lemma is equivalent to $(\exists N: \ rsem(N, S, pp, SQ)) \Rightarrow wtr(SQ, pp)$ and this implication follows from lemma *sem-rsem* together with lemma *well-typed poststates*.

QED: well-typed-rres.

Lemma 8 (well-typed-res-induct)**.** *For all naturals k, it holds:*

$$wtp(S, c) \wedge rssem_k(N, S, c, SQ) \Rightarrow wtr(SQ, c)$$

Proof:
By computational induction it is easy to show that for all k:

$$rssem_k(N, S, c, SQ) \Rightarrow rssem(N, S, c, SQ)$$

Let's denote this implication by $(+)$. We derive:

$$
\begin{array}{ll}
 & wtp(S, c) \wedge rssem_k(N, S, c, SQ) \\
\Rightarrow & \text{[[(+)]]} \\
 & wtp(S, c) \wedge rssem(N, S, c, SQ) \\
\Rightarrow & \text{[[definition of } rsem \text{]]} \\
 & rsem(N, S, c, SQ) \\
\Rightarrow & \text{[[lemma } well\text{-}typed\text{-}rres \text{]]} \\
 & wtr(SQ, c)
\end{array}
$$

QED: well-typed-res-induct.

4 The Logic

The programming logic presented in this section is based on four design goals:

1. It should be simple to use.
2. It should work on unspecified programs.
3. It should allow to express properties of the whole object store.
4. It should support the verification of extensible programs.

Since this report focusses on the soundness and completeness proof, we do not discuss the design goals and how they are met in detail. To achieve the first goal, we built on the experiences with Hoare logic. According to the second goal, the logic should allow to derive properties of programs that come without specification. Although this is the classical approach of Hoare logic, it is different from other approaches to the verification of object-oriented programs where the logic assumes specified programs (see e.g. [5]). The third design goal has lead to the decision to separate the rules for the object store from the rules that allow to reason about the dynamic program behavior. This way, these two aspects become orthogonal. Here, we only have to deal with the latter rules.

The fourth goal is the basis for modular verification. In contrast to procedural programs, object-oriented programs can have effects that are beyond the program text. For example, due to dynamic binding, a method m can invoke other methods that modify instance variables of fields that are declared outside the scope of m. There a two ways to handle this problem:

- By using a scope dependent semantics for the logical formulas.
- By using a scope independent semantics.

We follow the second approach. The main consequence for the programming logic is that it has to support abstraction over the state. Abstraction is supported by embedding the state formalization into a higher-order specification framework that supports the definition of new types and functions. These types and functions can be used to express abstraction functions and relations. They may be used in formulas of the Hoare logic.

An important property of the programming logic is that it supports program extensions: The proof of a triple done within the scope of one program Π remains valid in all well-formed programs extending Π by additional type declarations. This property is a prerequisite for modular program verification (although more is needed to make verification really modular) and reflects ordinary programming techniques based on reuse.

As in classical logic, we present the programming logic in three steps: What are formulas, what is their semantics, what are the proof rules. The programming logic is considered an extension of a general specification and verification framework (*logical framework* for short) such as Isabelle/HOL [10] or PVS [2]. We assume that the semantic definitions given in the previous sections can be and are expressed in this framework. The corresponding theory is called the language theory. In addition, each program Π gives rise to a program theory that contains constant definitions for all field identifiers declared in Π (of sort *FieldId*) and for all class and interface type identifiers (of sort *CTypeId* and *ITypeId*, resp.).

4.1 Formulas

Formulas of the programming logic are formulas of the underlying logical framework as well as sequents. A sequent consists of a set of assumptions and a triple that relates properties of pre- and poststates of program parts. Before we explain how sequents and triples are formed, we describe how type, field, and variable identifiers of programs can be used in formulas, in particular in pre- and postconditions. Essentially, we introduce constant symbols for these entities. More

precisely, a given Java-KE program Π defines a set Σ that contains all sort[5] and function[6] symbols as described in Sect. 2.2. These are exactly those symbols that occur in the underlying data and store model, including the program-dependent model parts, i.e. constant symbols for the types and fields declared in Π. We call Σ the signature of Π.

Furthermore, we treat parameters, program variables, and the variable \$ for the current object store syntactically as constant symbols of sort *Value* and *Store* respectively to simplify quantification and substitution rules as well as context conditions for pre- and postconditions. These symbols do not belong to Σ.[7] As it is common practice in Hoare logic, a variable v occurring in a pre- or postcondition represents the value of v in the prestate or poststate. If we want to refer to the corresponding variable identifier of sort *VarId*, we add a prime as postfix, that is, we write v'; in particular, $\texttt{this}', \texttt{par}' \in \mathit{VarId}$. The prime is omitted when it is clear from the context that we refer to the variable identifier. For example, if S is a state, we write $S(v)$ instead of $S(v')$.

As already said in Sect. 2, we assume that *Int* and *Bool* are sorts of the logical framework with unary operators *UnOp* and binary operators *BinOp* having the semantics of the programming language. That is, a term of sort *Exp* as defined in Fig. 1 can be considered a term of the logical framework as well. To keep things as simple as possible, in the following we do not distinguish between terms of sort *Exp* and the corresponding terms of the logical framework. That is, we use a shallow embedding of the expressions into the logical framework. (For more details on how numeric types of programming languages can be embedded into HOL, cf. [13].)

A *basic formula* for Π is a first-order formula over $\Sigma \cup \mathit{VarId} \cup \{\$\}$. A *triple* has the form $\{\ \mathbf{P}\ \}\ pp\ \{\ \mathbf{Q}\ \}$ where $\mathbf{P}$ and $\mathbf{Q}$ are basic formulas and pp is a program part of Π. Such a triple is called a *statement annotation*, *implementation annotation*, or *method annotation* if pp is a statement, a declared method, or virtual method, respectively. Pre- and postconditions of statement annotations and preconditions of implementation or method annotations have to be basic formulas over $\Sigma \cup \mathit{vis}(pp) \cup \{\$\}$. Postconditions of implementation or method annotations have to be formulas over $\Sigma \cup \{\texttt{res}, \texttt{exc}, \$\}$; in particular, the formal method parameters `this` and `par` must not occur in postconditions. If a program variable v may occur in a pre- or postcondition $\mathbf{R}$ of pp, we say that v is *admissible* for $\mathbf{R}$.

To handle recursive methods, we use *sequents* of the form $\mathcal{A} \rhd \mathbf{A}$ where $\mathcal{A}$ is a finite set of method and implementation annotations and $\mathbf{A}$ is a triple. Triples in $\mathcal{A}$ are called *assumptions* of the sequent and $\mathbf{A}$ is called the *consequent* of the sequent.[8] For convenience, the assumptions are written as a comma separated list without duplicates. Intuitively, a sequent expresses the fact that we can prove a triple based on some assumptions about methods. On an informal level, we do not distiguish between a triple and the corresponding sequent without assumptions. (As we will see in the next subsection the formal distinction corresponds to an implicit quantification.)

A *formula* of the logic is a basic formula or a sequent.

4.2 Semantics of formulas

The semantics of basic formulas is given by the logical framework. The semantics of sequents is described by a syntactic embedding into the logical framework. That is, we consider a sequent an abbreviation of a formula in the logical framework. The embedding is based on the following notations. Let Π be a program with signature Σ, F be a formula or term over Σ and S be a state. Then, $F[S]$ denotes the formula or term in which each occurrence of a program variable v is substituted by $S(v')$; similar for \$. A Hoare triple $\{\mathbf{P}\}pp\{\mathbf{Q}\}$ is considered an abbreviation for

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

[5] Examples for sort symbols are: *FieldId*, *MethodId*, *CTypeId*, *ITypeId*, ...

[6] Examples for function symbols are: τ, *init*, *styp*, *rtyp*, *defdm*, *dm*, ...

[7] This syntactical distinction between logical variables and program variables is similar to the way global and local variables are handled in temporal logic (cf. e.g. [3], p. 233ff.).

[8] As we consider sequents as formulas with a semantics, we do not use the "$\vdash$"-symbol (that is often used to express an inference relation) to separate assumptions from the consequent.

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. The semantics of a sequent

$$\{\mathbf{P}_1\}m_1\{\mathbf{Q}_1\},\ldots,\{\mathbf{P}_l\}m_l\{\mathbf{Q}_l\} \mid\!\!\rhd \{\mathbf{P}\}c\{\mathbf{Q}\}$$

is given by:

$$\forall N : \{\mathbf{P}_1\}\ m_1\{\mathbf{Q}_1\}\ \wedge\ \ldots\ \wedge\ \{\mathbf{P}_l\}\ m_l\{\mathbf{Q}_l\} \Rightarrow \{\mathbf{P}\}c\{\mathbf{Q}\}$$

The following lemma shows that this sequent semantics is compatible with the operational semantics based on *sem*:

Lemma 9 (triple-semantics)**.** *If a sequent has no assumptions, its semantics is the operational semantics defined by sem:*

$$(\mid\!\!\rhd \{\mathbf{P}\}pp\{\mathbf{Q}\}) \Leftrightarrow \mathrm{UC}(\forall S, SQ : \mathbf{P}[S]\ \wedge\ sem(S, pp, SQ) \Rightarrow \mathbf{Q}[SQ])$$

Proof:

$$\begin{array}{ll} & \mid\!\!\rhd \{\mathbf{P}\}pp\{\mathbf{Q}\} \\ \Leftrightarrow & \text{[[definition of sequents]]} \\ & \forall N : \mathrm{UC}(\forall S, SQ : \mathbf{P}[S]\ \wedge\ rsem(N, S, pp, SQ) \Rightarrow \mathbf{Q}[SQ]) \\ \Leftrightarrow & \text{[[predicate logic]]} \\ & \mathrm{UC}(\forall S, SQ : \mathbf{P}[S]\ \wedge\ (\exists N : rsem(N, S, pp, SQ)) \Rightarrow \mathbf{Q}[SQ]) \\ \Leftrightarrow & \text{[[lemma } \textit{sem-rsem} \text{]]} \\ & \mathrm{UC}(\forall S, SQ : \mathbf{P}[S]\ \wedge\ sem(S, pp, SQ) \Rightarrow \mathbf{Q}[SQ]) \end{array}$$

QED.triple-semantics

4.3 Axioms and Rules

The axioms and rules of the programming logic consist of

- the axioms and rules of the logical framework including the theories specifying the sorts and functions of Sect. 2 and
- the axioms and rules of the Hoare logic as described below.

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; that is, the use of program variables in a pre- or postcondition $\mathbf{R}$ is restricted to the variables admissible for $\mathbf{R}$ (see above). The presentation starts with the axioms and rules for the different statements. Then, we give the axioms and rules that can be applied to all program parts.

The axioms for the different assignment statements are essentially as in classical Hoare logic. The differences are caused by the need to handle the operations on the object store and by the possibility of exceptions:

cast-axiom:

$$\mid\!\!\rhd \left\{ \begin{array}{l} (\tau(e) \preceq \mathrm{T}\ \wedge\ \mathbf{P}[e/x]) \quad \vee \\ (\tau(e) \not\preceq \mathrm{T}\ \wedge\ \mathbf{P}[\$\langle\texttt{CastExc}\rangle/\$, new(\$, \texttt{CastExc})/\texttt{exc}]) \end{array} \right\} x = (\mathrm{T})e;\ \{\ \mathbf{P}\ \}$$

field-read-axiom:

$$\mid\!\!\rhd \left\{ \begin{array}{l} (y \neq \texttt{null}\ \wedge\ \mathbf{P}[\$(y.a)/x]) \quad \vee \\ (y = \texttt{null}\ \wedge\ \mathbf{P}[\$\langle\texttt{NullPExc}\rangle/\$, new(\$, \texttt{NullPExc})/\texttt{exc}]) \end{array} \right\} x = y.a;\ \{\ \mathbf{P}\ \}$$

field-write-axiom:

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

new-axiom:

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

The rules for the compound statements are as in classical Hoare logic except that we have to take care of the exceptions:

seq-rule:

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

if-rule:

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

Note that `true` and `false` denote the elements of the term language, not the elements of Java.

while-rule:

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

catch-rule:

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

block-rule:

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

Note that the triple in the conclusion is only correctly formed if $\mathbf{P}$ and $\mathbf{Q}$ do not contain v. The variable v cannot be visible outside the block statement in Java, because Java does not allow hiding of variables defined in enclosing scopes.

From an object-oriented point of view, the most interesting rules are those that handle method invocation, inheritance, and subtyping. Method invocation is handled by two rules and the following axiom that handles the case that the target object is `null`:

invoc-exc:

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

The first rule for invocations allows to transfer the properties that have been proven for a virtual method *T:m* to the invocation site. The type T is the static type of the target expression. The context conditions of Java guarantee that method m is declared for T. As will be expressed by the subtype and class rules, properties of a virtual method *T:m* are those that hold for all subtype methods of *T:m*.

invoc-rule:

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

According to the context conditions for triples, **P** may only contain the program variables `this`, `par` and `exc`, and **Q** may only contain `res` and `exc` in instantiations of this rule. The following rule has to be used to show that program variables different from x and `exc` hold the same value in pre- and poststate, i.e. that a method invocation may only modify x and `exc` but no other program variables:

invoc-var-rule:

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

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

For calls to methods in superclasses, we provide almost the same rules. In these rules, $super_T$ denotes the actual type in which the method m is called if the Java code contains the method call `super.m(e)`. This type can be statically determined and thus already inserted into the program's abstract syntax tree by the static program analysis. Note that in the call rules there is no substitution on `this` because the receiver object is not changed by the super call.

call-rule:

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

call-var-rule:

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

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

The next three rules describe how properties of methods can be derived. The implementation rule allows to discard an implementation annotation from the assumptions of a sequent if the annotation holds for the body of the corresponding method, more precisely:

impl-rule:

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

The proof technique for virtual methods is based on the formalization of the subtype relation and two rules. As already explained in Subsect. 2.2, we assume that the subtype relation $\preceq$ of a program Π is completely axiomatized. In particular, we assume that we can derive for each type T a *characterizing formula* of the following form:

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

where $T_1, \ldots, T_n$ are the subtypes of T. Since the whole program is known at verification time (see the context conditions for Java-KE, Sec. **??**), all subtypes of a type are known as well. Thus, it is easy to state and prove these formulas. For modular verification, one has to distinguish between the known types for which verification can be done and the not yet known types for which certain properties are assumed and proved later. Here, we do not consider modular verification and refer the reader to Peter Müller's thesis [8] that develops modular verification techniques based on our programming logic.

The technique to derive a method annotation is based on two specific rules for virtual methods and several rules that work on all triples. We first explain how a triple of the form $\{\mathbf{P}\}\ T{:}m\ \{\mathbf{Q}\}$ is proved. Then, we explain the rules. By the swiss-rule (see below), it is allowed to assume correct typing of variables and parameters in the prestate, in particular for `this`. That is, it suffices to show

$$\{\tau(\texttt{this}) \preceq T \wedge \mathbf{P}\}\ T{:}m\ \{\mathbf{Q}\}$$

According to the characterizing formula this is equivalent to

$$\{(\; \tau(\texttt{this}) = T \vee \tau(\texttt{this}) \preceq T_1 \vee \cdots \vee \tau(\texttt{this}) \preceq T_n \;) \wedge \mathbf{P}\}\ T{:}m\ \{\mathbf{Q}\}$$

Using the disjunct-rule (see below), this goal can be derived from $\{\tau(\texttt{this}) = T \wedge \mathbf{P}\}\ T{:}m\ \{\mathbf{Q}\}$ and $\{\tau(\texttt{this}) \preceq T_i \wedge \mathbf{P}\}\ T{:}m\ \{\mathbf{Q}\}$ for $1 \le i \le n$. The class rule allows to derive the first triple:

class-rule:

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

Recall that $\mathit{defdm}(T, m)$ states that T is a class and m is a method implemented in or inherited by T. Also recall that for each class T in which m is inherited but not implemented, $\mathit{dm}(T, m)$ yields the same representative.

The remaining goal triples are proved by the subtype rule:

subtype-rule:

$$\frac{\begin{array}{l} S \prec T \\ \mathcal{A} \mid\!\!\rhd\ \{\ \mathbf{P}\ \}\ S{:}m\ \{\ \mathbf{Q}\ \} \end{array}}{\mathcal{A} \mid\!\!\rhd\ \{\ \tau(\texttt{this}) \preceq S\ \wedge\ \mathbf{P}\ \}\ T{:}m\ \{\ \mathbf{Q}\ \}}$$

The above axioms and rules capture the semantics of specific constructs of Java-KE. The following axioms and rules provide the reasoning capabilities that are independent of the underlying program parts. As above $\mathcal{A}$ stands for a set of assumptions and **P**, **PP**, **P1**, **P2**, **Q**, **QQ**, **Q1**, **Q2** are formulas adhering to the restrictions of the corresponding triples; **A** stands for syntactically correct method or implementation annotations; Y and Z denote distinct logical variables of the same sort, and *pp* stands for program parts.

assumpt-axiom:

$$\mathbf{A} \mid\!\rhd \mathbf{A}$$

false-axiom:

$$\mid\!\rhd \{\ \text{FALSE}\ \}\ pp\ \{\ \text{FALSE}\ \}$$

assumpt-intro-rule:

$$\frac{\mathcal{A} \mid\!\rhd \mathbf{A}}{\mathcal{A}, \mathbf{A}_0 \mid\!\rhd \mathbf{A}}$$

assumpt-elim-rule:

$$\frac{\mathcal{A} \mid\!\rhd \mathbf{A}_0 \quad \mathcal{A}, \mathbf{A}_0 \mid\!\rhd \mathbf{A}}{\mathcal{A} \mid\!\rhd \mathbf{A}}$$

disjunct-rule:

$$\frac{\mathcal{A} \mid\!\rhd \{\ \mathbf{P1}\ \}\ pp\ \{\ \mathbf{Q1}\ \} \quad \mathcal{A} \mid\!\rhd \{\ \mathbf{P2}\ \}\ pp\ \{\ \mathbf{Q2}\ \}}{\mathcal{A} \mid\!\rhd \{\ \mathbf{P1} \vee \mathbf{P2}\ \}\ pp\ \{\ \mathbf{Q1} \vee \mathbf{Q2}\ \}}$$

ex-rule:

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

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 $\mathbf{Q}$.

Finally, the logic supports a multi-purpose rule, the so-called swiss-rule (like a swiss army knife). The swiss-rule allows to

- **S**trengthen preconditions,
- **W**eaken postconditions,
- add **I**nvariants to pre- and postconditions,
- **S**ubstitute free variables by terms, and
- **S**upply the pre- and postconditions with typing and other information.

Before we discuss the application of the rule, we explain its syntactical aspects:

swiss-rule:

$$\frac{\begin{array}{l}\mathbf{PP} \wedge \tau(v) \preceq \mathit{styp}(v', pp) \wedge \texttt{this} \neq \texttt{null} \wedge \texttt{exc} = \texttt{null} \wedge \mathit{wts}(\$) \Rightarrow \mathbf{P} \\ \mathcal{A} \mid\!\rhd \{\ \mathbf{P}\ \}\ pp\ \{\ \mathbf{Q}\ \} \\ \mathbf{Q} \ \wedge\ \tau(w) \preceq \mathit{styp}(w', pp) \ \wedge\ \texttt{this} \neq \texttt{null} \ \wedge\ \mathit{wts}(\$) \Rightarrow \mathbf{QQ}\end{array}}{\mathcal{A} \mid\!\rhd \{\ \mathbf{PP}[t/Z] \ \wedge\ \mathbf{R}\ \}\ pp\ \{\ \mathbf{QQ}[t/Z] \ \wedge\ \mathbf{R}\ \}}$$

where
v program variable, admissible in pre(pp),
w program variable, admissible in post(pp),
t Σ-term,
Z logical variable of same sort as t,
$\mathbf{R}$ Σ-formula.

Recall that Σ does not contain the program variables or \$. That is, t and $\mathbf{R}$ do not depend on the state. That is why the swiss-rule allows to derive $\{\mathbf{P} \wedge \mathbf{R}\} pp \{\mathbf{Q} \wedge \mathbf{R}\}$ and $\{\mathbf{P}[t/Z]\} pp \{\mathbf{Q}[t/Z]\}$ from $\{\mathbf{P}\} pp \{\mathbf{Q}\}$. Strengthening is supported by using the instantiation $\mathbf{QQ} \equiv \mathbf{Q},\ t \equiv Z,\ \mathbf{R} \equiv \text{TRUE}$ (weakening is done similarly). By using the instantiations

$$\begin{array}{rcl}\mathbf{P} & \equiv & \mathbf{PP} \wedge \tau(v) \preceq \mathit{styp}(v', pp) \wedge \texttt{this} \neq \texttt{null} \wedge \texttt{exc} = \texttt{null} \wedge \mathit{wts}(\$) \\ \mathbf{QQ} & \equiv & \mathbf{Q} \ \wedge\ \tau(w) \preceq \mathit{styp}(w', pp) \ \wedge\ \texttt{this} \neq \texttt{null} \ \wedge\ \mathit{wts}(\$)\end{array}$$

one can eliminate typing conjuncts and properties of `this` and `exc` in the precondition or add typing conjuncts to the postcondition.

In the two implications, the additional conjuncts provide the user with information that is known to always hold semantically in the precondition and postcondition, respectively. The swiss rule provides the means to add this semantic information to the current proof goal for free.

The two implications seem to indicate that this rule allows to strengthen and weaken to arbitrary formulae if one of the conjuncts is false. But since the first implication must hold for a fixed **P** and **PP** (for the second implication, for a fixed **Q** and **QQ**, correspondingly) in all interpretations, especially in those where all added conjuncts yield true. Thus, this rule does not produce unsoundness.

The swiss-rule takes care of adaptation completeness [4] since it allows to rename logical variables. This renaming mechanism has been presented in [11] already. Kleymann performs the substitution directly on the strengthening and weakening implications, which makes his solution less elegant and simple.

To clarify matters, we show the four rules that can be derived from the swiss-rule:

strength-rule:

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

weak-rule:

$$\frac{\mathcal{A} \mid\!\rhd \{\ \mathbf{P}\ \}\ \mathit{pp}\ \{\ \mathbf{Q}\ \} \quad \mathbf{Q} \ \wedge\ \tau(w) \preceq \mathit{styp}(w', \mathit{pp}) \ \wedge\ \texttt{this} \neq \texttt{null} \ \wedge\ \mathit{wts}(\$) \Rightarrow \mathbf{QQ}}{\mathcal{A} \mid\!\rhd \{\ \mathbf{P}\ \}\ \mathit{pp}\ \{\ \mathbf{QQ}\ \}}$$

invar-rule:

$$\frac{\mathcal{A} \mid\!\rhd \{\ \mathbf{P}\ \}\ \mathit{pp}\ \{\ \mathbf{Q}\ \}}{\mathcal{A} \mid\!\rhd \{\ \mathbf{P} \wedge \mathbf{R}\ \}\ \mathit{pp}\ \{\ \mathbf{Q} \wedge \mathbf{R}\ \}}$$

subst-rule:

$$\frac{\mathcal{A} \mid\!\rhd \{\ \mathbf{P}\ \}\ \mathit{pp}\ \{\ \mathbf{Q}\ \}}{\mathcal{A} \mid\!\rhd \{\ \mathbf{P}[t/Z]\ \}\ \mathit{pp}\ \{\ \mathbf{Q}[t/Z]\ \}}$$

These four rules can be used to "derive" the swiss-rule as follows:

$$\begin{aligned}
\mathbf{R}_1 &= \tau(\texttt{this}) \preceq \mathit{styp}(\texttt{this}', \mathit{pp}) \wedge \texttt{this} \neq \texttt{null} \wedge \texttt{exc} = \texttt{null} \wedge \mathit{wts}(\$) \\
\mathbf{R}_2 &= \tau(\texttt{this}) \preceq \mathit{styp}(\texttt{this}', \mathit{pp}) \ \wedge\ \texttt{this} \neq \texttt{null} \ \wedge\ \mathit{wts}(\$)
\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{A} \mid\!\rhd \{\ \mathbf{P}\ \}\ \mathit{pp}\ \{\ \mathbf{Q}\ \} \qquad \mathbf{PP} \wedge \mathbf{R}_1 \Rightarrow \mathbf{P}}{\mathcal{A} \mid\!\rhd \{\ \mathbf{PP}\ \}\ \mathit{pp}\ \{\ \mathbf{Q}\ \}}\,[\mathit{strength}] \qquad \mathbf{Q} \ \wedge\ \mathbf{R}_2 \Rightarrow \mathbf{QQ}}{\mathcal{A} \mid\!\rhd \{\ \mathbf{PP}\ \}\ \mathit{pp}\ \{\ \mathbf{QQ}\ \}}\,[\mathit{weak}]}{\mathcal{A} \mid\!\rhd \{\ \mathbf{PP}[t/Z]\ \}\ \mathit{pp}\ \{\ \mathbf{QQ}[t/Z]\ \}}\,[\mathit{subst}]}{\mathcal{A} \mid\!\rhd \{\ \mathbf{PP}[t/Z] \wedge \mathbf{R}\ \}\ \mathit{pp}\ \{\ \mathbf{QQ}[t/Z] \wedge \mathbf{R}\ \}}\,[\mathit{invar}]$$

For practical program verification, in particular modular verification, the following complementary axiom and rules are helpful. However, as they are not needed for completeness, they will not be considered in the rest of this report:

conjunct-rule:

$$\frac{\mathcal{A} \mid\!\rhd \{\ \mathbf{P}1\ \}\ \mathit{pp}\ \{\ \mathbf{Q}1\ \} \quad \mathcal{A} \mid\!\rhd \{\ \mathbf{P}2\ \}\ \mathit{pp}\ \{\ \mathbf{Q}2\ \}}{\mathcal{A} \mid\!\rhd \{\ \mathbf{P}1 \ \wedge\ \mathbf{P}2\ \}\ \mathit{pp}\ \{\ \mathbf{Q}1 \ \wedge\ \mathbf{Q}2\ \}}$$

all-rule:

$$\frac{\mathcal{A} \mid\!\rhd \{\ \mathbf{P}[Y/Z]\ \}\ \mathit{pp}\ \{\ \mathbf{Q}\ \}}{\mathcal{A} \mid\!\rhd \{\ \mathbf{P}[Y/Z]\ \}\ \mathit{pp}\ \{\ \forall Z : \mathbf{Q}\ \}}$$

true-axiom:

$$\mid\!\rhd \{\ \mathrm{TRUE}\ \}\ \mathit{pp}\ \{\ \mathrm{TRUE}\ \}$$

5 Soundness Proof

In this section, we show that each axiom and rule is sound w.r.t. the semantics. Axioms will be derived from the semantics. For rules, it will be shown that the premises imply the conclusion. In these proofs, we use the following two lemmas:

Lemma 10 (substitution)**.** *Let* $\mathbf{F}$ *be a basic formula or term,* $x_1, \ldots, x_n$ *be program variables or* \$*, and* $t_1, \ldots, t_n$ *ground terms of suitable sort, that is, terms without logical variables. Then, it holds for every state* S*:*

$$(\mathbf{F}[t_1/x_1, \ldots, t_n/x_n])[S] \quad \Leftrightarrow \quad \mathbf{F}[S[x_1 := S(t_1), \ldots, x_n := S(t_n)]]$$

Proof:
A complete proof of the lemma would be done by induction on the structure of $\mathbf{F}$. We consider here only the interesting base case, namely that $\mathbf{F} \equiv y$ for some program variable y. That is, we have to show:

$$(y[t_1/x_1, \ldots, t_n/x_n])[t_1/x_1, \ldots, t_n/x_n][S] \quad = \quad y[S[x_1 := S(t_1), \ldots, x_n := S(t_n)]]$$

This is proven by a simple case distinction:

Case a: There is an i such that $y \equiv x_i$:

$$\begin{array}{ll} & (y[t_1/x_1, \ldots, t_n/x_n])[S] \\ = & \\ & (x_i[t_1/x_1, \ldots, t_n/x_n])[S] \\ = & \\ & t_i[S] \\ = & \\ & S(t_i) \\ = & \\ & (S[x_1 := S(t_1), \ldots, x_n := S(t_n)])(x_1) \\ = & \\ & x_i[S[x_1 := S(t_1), \ldots, x_n := S(t_n)]] \\ = & \\ & y[S[x_1 := S(t_1), \ldots, x_n := S(t_n)]] \end{array}$$

Case b: There is no i such that $y \equiv x_i$:

$$\begin{array}{ll} & (y[t_1/x_1, \ldots, t_n/x_n])[S] \\ = & \\ & y[S] \\ = & \\ & S(y) \\ = & \\ & (S[x_1 := S(t_1), \ldots, x_n := S(t_n)])(y) \\ = & \\ & y[S[x_1 := S(t_1), \ldots, x_n := S(t_n)]] \end{array}$$

QED: substitution.

Lemma 11 (quantification simplification, quasi)**.** *To prove the soundness of a rule of the form:*

$$\frac{\begin{array}{l} \mathcal{A} \models \ \{\ \mathbf{P}_1\ \}\, s_1\, \{\ \mathbf{Q}_1\ \} \\ \ldots \\ \mathcal{A} \models \ \{\ \mathbf{P}_m\ \}\, s_m\, \{\ \mathbf{Q}_m\ \} \end{array}}{\mathcal{A} \models \ \{\ \mathbf{P}\ \}\, s\, \{\ \mathbf{Q}\ \}}$$

it suffices to show one of the following two implications:

$$
\begin{array}{ll}
 & (\forall S, SQ : \mathbf{P}_1[S] \ \wedge \ rsem(N, S, s1, SQ) \Rightarrow \mathbf{Q}_1[SQ]) \\
\wedge & \ldots \\
\wedge & (\forall S, SQ : \mathbf{P}_m[S] \ \wedge \ rsem(N, S, s1, SQ) \Rightarrow \mathbf{Q}_m[SQ]) \\
\Rightarrow & \mathbf{P}[S] \ \wedge \ rsem(N, S, s, SQ) \Rightarrow \mathbf{Q}[SQ]
\end{array}
$$

or

$$
\begin{array}{ll}
 & \mathrm{UC}(\forall S, SQ : \mathbf{P}_1[S] \ \wedge \ rsem(N, S, s_1, SQ) \Rightarrow \mathbf{Q}_1[SQ]) \\
\wedge & \ldots \\
\wedge & \mathrm{UC}(\forall S, SQ : \mathbf{P}_m[S] \ \wedge \ rsem(N, S, s_m, SQ) \Rightarrow \mathbf{Q}_m[SQ]) \\
\Rightarrow & \mathbf{P}[S] \ \wedge \ rsem(N, S, s, SQ) \Rightarrow \mathbf{Q}[SQ]
\end{array}
$$

where UC *is defined as in Subsect. 4.2 and where we assume w.l.g. that the free variables different from N in the m conjuncts of the premise and the conclusion are renamed in such a way that they are pairwise disjoint and different from N.*

Note that within the implications, the free variable N is the same for all conjuncts and the conclusion, whereas, according to the semantics of sequents, it is bound in the antecedents and consequents of the rule (cf. Subsect. 4.2). Furthermore, the implications do not refer to the assumptions of the sequents.

Proof:
The proof uses two facts from predicate logic (PL):

Fact 1: Let $\mathbf{P}$ and $\mathbf{Q}$ be basic formulas such that X does not occur free in $\mathbf{Q}$ and Y does not occur free in $\mathbf{P}$. If $\mathbf{P} \Rightarrow \mathbf{Q}$ is valid, then $(\forall X : \mathbf{P}) \Rightarrow \mathbf{Q}$ and $(\forall X : \mathbf{P}) \Rightarrow (\forall Y : \mathbf{Q})$ are valid as well.
Fact 2: Let $\mathbf{P}$ and $\mathbf{Q}$ be basic formulas. If $\mathbf{P} \Rightarrow \mathbf{Q}$ is valid, then $(\forall X : \mathbf{P}) \Rightarrow (\forall X : \mathbf{Q})$ is valid as well.

Let AP denote the conjunction of the semantic embeddings of the assumptions in $\mathcal{A}$. We get:

$$
\begin{array}{ll}
 & (\forall S, SQ : \mathbf{P}_1[S] \ \wedge \ rsem(N, S, s_1, SQ) \Rightarrow \mathbf{Q}_1[SQ]) \\
\wedge & \ldots \\
\wedge & (\forall S, SQ : \mathbf{P}_m[S] \ \wedge \ rsem(N, S, s_m, SQ) \Rightarrow \mathbf{Q}_m[SQ]) \\
\Rightarrow & \mathbf{P}[S] \ \wedge \ rsem(N, S, s, SQ) \Rightarrow \mathbf{Q}[SQ] \\
\Rightarrow & [\![\text{ fact 1; predicate logic; free variables in conjuncts and conclusion are pairwise disjoint}]\!] \\
 & \mathrm{UC}(\forall S, SQ : \mathbf{P}_1[S] \ \wedge \ rsem(N, S, s_1, SQ) \Rightarrow \mathbf{Q}_1[SQ]) \\
\wedge & \ldots \\
\wedge & \mathrm{UC}(\forall S, SQ : \mathbf{P}_m[S] \ \wedge \ rsem(N, S, s_m, SQ) \Rightarrow \mathbf{Q}_m[SQ]) \\
\Rightarrow & \mathrm{UC}(\mathbf{P}[S] \ \wedge \ rsem(N, S, s, SQ) \Rightarrow \mathbf{Q}[SQ]) \\
\Rightarrow & [\![\text{ predicate logic }]\!] \\
 & (AP \Rightarrow \mathrm{UC}(\forall S, SQ : \mathbf{P}_1[S] \ \wedge \ rsem(N, S, s_1, SQ) \Rightarrow \mathbf{Q}_1[SQ])) \\
\wedge & \ldots \\
\wedge & (AP \Rightarrow \mathrm{UC}(\forall S, SQ : \mathbf{P}_m[S] \ \wedge \ rsem(N, S, s_m, SQ) \Rightarrow \mathbf{Q}_m[SQ])) \\
\Rightarrow & (AP \Rightarrow \mathrm{UC}(\mathbf{P}[S] \ \wedge \ rsem(N, S, s, SQ) \Rightarrow \mathbf{Q}[SQ])) \\
\Rightarrow & [\![\text{ fact 2 and predicate logic }]\!] \\
 & (\forall N : \ AP \Rightarrow \mathrm{UC}(\forall S, SQ : \mathbf{P}_1[S] \ \wedge \ rsem(N, S, s_1, SQ) \Rightarrow \mathbf{Q}_1[SQ])) \\
\wedge & \ldots \\
\wedge & (\forall N : \ AP \Rightarrow \mathrm{UC}(\forall S, SQ : \mathbf{P}_m[S] \ \wedge \ rsem(N, S, s_m, SQ) \Rightarrow \mathbf{Q}_m[SQ])) \\
\Rightarrow & (\forall N : \ AP \Rightarrow \mathrm{UC}(\mathbf{P}[S] \ \wedge \ rsem(N, S, s, SQ) \Rightarrow \mathbf{Q}[SQ]))
\end{array}
$$

The last implication corresponds exactly to the semantical embedding of the rule.
QED: quasi.

In the following, we consider the axioms and rules in turn and show based on the operational semantics that each axiom is valid and that the antecedents of a rule imply the succedent.

Soundness of the cast-axiom:
For all N, for all free logical variables in $\mathbf{P}$, and for all states S and SQ, we have to show:

$$((\tau(e) \preceq T \wedge \mathbf{P}[e/x]) \vee (\tau(e) \not\preceq T \wedge \mathbf{P}[\$\langle \texttt{CastExc} \rangle / \$, new(\$, \texttt{CastExc})/\texttt{exc}]))[S] \\ \wedge\ rsem(N, S, c, SQ) \Rightarrow \mathbf{P}[SQ]$$

with $c \equiv$ "$x = (\mathrm{T})e;$". According to the substitution lemma, this is equivalent to:

$$((\tau(S(e)) \preceq T \wedge \mathbf{P}[S[x := S(e)]]) \vee (\tau(S(e)) \not\preceq T \wedge \mathbf{P}[S[\$:= S(\$)\langle \texttt{CastExc} \rangle, \texttt{exc} := new(S(\$), \texttt{CastExc})]])) \\ \wedge\ rsem(N, S, c, SQ) \ \Rightarrow\ \mathbf{P}[SQ]$$

From the definition of *rssem* (see Sect. 3), we get the following two semantic properties:

$$rssem(N, S, c, SQ) \wedge \tau(S(e)) \preceq T \ \Rightarrow\ SQ = S[x := S(e)]$$
$$rssem(N, S, c, SQ) \wedge \tau(S(e)) \not\preceq T \ \Rightarrow\ SQ = S[\$:= S(\$)\langle \texttt{CastExc} \rangle, \texttt{exc} := new(S(\$), \texttt{CastExc})]$$

These properties imply what we have to show.
QED: cast-axiom.

Soundness of the field-read-axiom:
For all N, for all free logical variables in $\mathbf{P}$, and for all states S and SQ, we have to show:

$$((y \neq \texttt{null} \ \wedge\ \mathbf{P}[\$(y.a)/x]) \ \vee\ (y = \texttt{null} \ \wedge\ \mathbf{P}[\$\langle \texttt{NullPExc} \rangle / \$, new(\$, \texttt{NullPExc})/\texttt{exc}]))[S] \\ \wedge\ rsem(N, S, c, SQ) \Rightarrow \mathbf{P}[SQ]$$

with $c \equiv$ "$x = y.a;$". According to the substitution lemma, this is equivalent to:

$$((S(y) \neq \texttt{null} \wedge \mathbf{P}[S[x := S(\$(y.a))]]) \\ \vee\ (S(y) = \texttt{null} \wedge \mathbf{P}[S[\$:= S(\$)\langle \texttt{NullPExc} \rangle, \texttt{exc} := new(S(\$), \texttt{NullPExc}]]])) \\ \wedge\ rsem(N, S, c, SQ) \Rightarrow \mathbf{P}[SQ]$$

From the definition of *rssem* (see Sect. 3), we get the following two semantic properties:

$$rssem(N, S, c, SQ) \wedge S(y) \neq \texttt{null} \ \Rightarrow\ SQ = S[x := S(\$)(S(y).a)]$$
$$rssem(N, S, c, SQ) \wedge S(y) = \texttt{null} \ \Rightarrow \\ SQ = S[\$:= S(\$)\langle \texttt{NullPExc} \rangle, \texttt{exc} := new(S(\$), \texttt{NullPExc})]$$

These properties imply what we have to show.
QED: field-read-axiom.

Soundness of the field-write-axiom:
For all N, for all free logical variables in $\mathbf{P}$, and for all states S and SQ, we have to show:

$$((x \neq \texttt{null} \wedge \mathbf{P}[\$\langle x.a := e \rangle / \$]) \ \vee\ (x = \texttt{null} \wedge \mathbf{P}[\$\langle \texttt{NullPExc} \rangle / \$, new(\$, \texttt{NullPExc})/\texttt{exc}]))[S] \\ \wedge\ rsem(N, S, c, SQ) \Rightarrow \mathbf{P}[SQ]$$

with $c \equiv$ "$x.a = e;$". According to the substitution lemma, this is equivalent to:

$$((S(x) \neq \texttt{null} \wedge \mathbf{P}[S(\$)\langle x.a := S(e) \rangle]) \\ \vee\ (S(x) = \texttt{null} \wedge \mathbf{P}[S[\$:= S(\$)\langle \texttt{NullPExc} \rangle, \texttt{exc} := new(S(\$), \texttt{NullPExc})]])) \\ \wedge\ rsem(N, S, c, SQ) \Rightarrow \mathbf{P}[SQ]$$

From the definition of *rssem* (see Sect. 3), we get the following two semantic properties:

$$rssem(N, S, c, SQ) \wedge S(x) \neq \texttt{null} \ \Rightarrow\ SQ = S[\$:= S(\$)\langle x.a := e \rangle]$$
$$rssem(N, S, c, SQ) \wedge S(x) = \texttt{null} \ \Rightarrow \\ SQ = S[\$:= S(\$)\langle \texttt{NullPExc} \rangle, \texttt{exc} := new(S(\$), \texttt{NullPExc})]$$

These properties imply what we have to show.
QED: field-write-axiom.

Soundness of the new-axiom:
For all N, for all free logical variables in $\mathbf{P}$, and for all states S and SQ, we have to show:

$$\mathbf{P}[new(\$, T)/x, \$\langle T\rangle/\$] \wedge rsem(N, S, c, SQ) \Rightarrow \mathbf{P}[SQ]$$

with $c \equiv$ "$x = \texttt{new}\ T();$". According to the substitution lemma, this is equivalent to:

$$\mathbf{P}[S[x := new(S(\$), T), \$:= S(\$)\langle T\rangle]] \wedge rsem(N, S, c, SQ) \Rightarrow \mathbf{P}[SQ]$$

From the definition of *rssem* (see Sect. 3), we get the following semantic propertiy:

$$rssem(N, S, c, SQ) \ \Rightarrow \ SQ = S[x := new(S(\$), T), \$:= S(\$)\langle T\rangle]$$

These properties imply what we have to show.
QED: new-axiom.

Soundness of the seq-rule:
Let $c \equiv$ "$s_1\ s_2$". According to lemma *quasi*, it suffices to show for all N, for all free logical variables in $\mathbf{P}$, $\mathbf{Q}$, and $\mathbf{R}$, and for all states S and SQ:

$$\begin{aligned}
&(\forall S_P, S_R : \mathbf{P}[S_P] \ \wedge \ rsem(N, S_P, s_1, S_R) \\
&\qquad\qquad \Rightarrow \ \ (S_R(\texttt{exc}) \neq \texttt{null} \wedge \mathbf{Q}[S_R]) \ \vee \ (S_R(\texttt{exc}) = \texttt{null} \wedge \mathbf{R}[S_R])\ ) \qquad (1)\\
&\wedge\ (\forall S_R, S_Q : \mathbf{R}[S_R] \ \wedge \ rsem(N, S_R, s_2, S_Q) \Rightarrow \mathbf{Q}[S_Q]) \qquad (2)\\
&\Rightarrow (\mathbf{P}[S] \ \wedge \ rsem(N, S, s_1\ s_2, SQ) \Rightarrow \mathbf{Q}[SQ])
\end{aligned}$$

From the definition of *rssem* (see Sect. 3), we get the following semantic property:

$$\begin{aligned}
&rssem(N, S, s_1\ s_2, SQ) \Leftrightarrow \\
&\quad (\exists SR : rssem(N, S, s_1, SR) \wedge SR(\texttt{exc}) = \texttt{null} \wedge rssem(N, SR, s_2, SQ)) \qquad (3)\\
&\vee \quad (rssem(N, S, s_1, SQ) \wedge SQ(\texttt{exc}) \neq \texttt{null})
\end{aligned}$$

Additionally, we know

$$wtp(S, c) = wtp(S, s_1) \wedge wtp(S, s_2) \qquad (4)$$

because $c \equiv$ "$s_1\ s_2$" and $vis(c) = vis(s_1)$, and from this we get

$$\begin{aligned}
&\quad rsem(N, S, s_1, SR) \wedge SR(\texttt{exc}) = \texttt{null}\\
\Longrightarrow\ &[\![\text{lemma 7: } \textit{well_typed_rres}]\!]\\
&\quad wtr(SR, s_1) \wedge SR(\texttt{exc}) = \texttt{null}\\
\Longrightarrow\ &[\![\text{def. of } wtr]\!]\\
&\quad wt(SR, s_1) \wedge SR(\texttt{exc}) = \texttt{null}\\
\Longrightarrow\ &[\![\text{def. of } wtp]\!] \qquad (5)\\
&\quad wtp(SR, s_1)\\
\Longrightarrow\ &[\![(4)]\!]\\
&\quad wtp(SR, s_1 s_2)\\
\Longrightarrow\ &[\![(4)]\!]\\
&\quad wtp(SR, s_2)
\end{aligned}$$

We derive $\mathbf{Q}[SQ]$ from $\mathbf{P}[S] \ \wedge \ rsem(N, S, c, SQ)$ using the premises of the rule and the semantic property in the derivation:

$$\mathbf{P}[S] \wedge rsem(N, S, c, SQ)$$

$\Longrightarrow$ [[definition of *rsem*]]
$\mathbf{P}[S] \wedge wtp(S,c) \wedge rssem(N,S,c,SQ)$
$\Longrightarrow$ [[(4)]]
$\mathbf{P}[S] \wedge wtp(S,s_1) \wedge rssem(N,S,c,SQ)$
$\Longrightarrow$ [[semantical property (3)]]
$\mathbf{P}[S] \wedge wtp(S,s_1)$
$\wedge ((\exists SR : rssem(N,S,s_1,SR) \wedge SR(\texttt{exc}) = \texttt{null} \wedge rssem(N,SR,s_2,SQ))$
$\vee (rssem(N,S,s_1,SQ) \wedge SQ(\texttt{exc}) \neq \texttt{null})$)
$\Longrightarrow$ [[predicate logic]]
$(\exists SR : \mathbf{P}[S] \wedge wtp(S,s_1) \wedge rssem(N,S,s_1,SR) \wedge SR(\texttt{exc}) = \texttt{null} \wedge rssem(N,SR,s_2,SQ)$
$\vee (\mathbf{P}[S] \wedge wtp(S,s_1) \wedge rssem(N,S,s_1,SQ) \wedge SQ(\texttt{exc}) \neq \texttt{null}$)
$\Longrightarrow$ [[definition of *rsem*]]
$(\exists SR : \mathbf{P}[S] \wedge rsem(N,S,s_1,SR) \wedge SR(\texttt{exc}) = \texttt{null} \wedge rssem(N,SR,s_2,SQ))$
$\vee (\mathbf{P}[S] \wedge rsem(N,S,s_1,SQ) \wedge SQ(\texttt{exc}) \neq \texttt{null}$)
$\Longrightarrow$ [[apply (1) on both disjuncts]]
$(\exists SR : ((SR(\texttt{exc}) \neq \texttt{null} \wedge \mathbf{Q}[SR]) \vee (SR(\texttt{exc}) = \texttt{null} \wedge \mathbf{R}[SR]))$
$\wedge rsem(N,S,s_1,SR) \wedge SR(\texttt{exc}) = \texttt{null} \wedge rssem(N,SR,s_2,SQ))$
$\vee (((SQ(\texttt{exc}) \neq \texttt{null} \wedge \mathbf{Q}[SQ]) \vee (SQ(\texttt{exc}) = \texttt{null} \wedge \mathbf{R}[SQ]) \wedge SQ(\texttt{exc}) \neq \texttt{null}$)
$\Longrightarrow$ [[predicate logic]]
$(\exists SR : \mathbf{R}[SR] \wedge rsem(N,S,s_1,SR) \wedge SR(\texttt{exc}) = \texttt{null} \wedge rssem(N,SR,s_2,SQ))$
$\vee \mathbf{Q}[SQ]$
$\Longrightarrow$ [[(5)]]
$(\exists SR : \mathbf{R}[SR] \wedge wtp(SR,s_2) \wedge rssem(N,SR,s_2,SQ)) \vee \mathbf{Q}[SQ]$
$\Longrightarrow$ [[definition of *rsem*]]
$(\exists SR : \mathbf{R}[SR] \wedge rsem(N,SR,s_2,SQ)) \vee \mathbf{Q}[SQ]$
$\Longrightarrow$ [[apply 2. antecedent to first disjunct]]
$(\exists SR : \mathbf{Q}[SQ]) \vee \mathbf{Q}[SQ]$
$\Longrightarrow$ [[SR not free in $\mathbf{Q}[SQ]$]]
$\mathbf{Q}[SQ]$

QED seq-rule.

Soundness of the if-rule:

Let $c \equiv$ "$\texttt{if}(e)\{s_1\}\texttt{else}\{s_2\}$". According to lemma *quasi*, it suffices to show for all N, for all free logical variables in $\mathbf{P}$ and $\mathbf{Q}$, and for all states S and SQ:

$$(\forall S_P, S_Q : (e \wedge \mathbf{P})[S_P] \wedge rsem(N,S_P,s_1,S_Q) \Rightarrow \mathbf{Q}[S_Q]) \tag{1}$$
$$\wedge (\forall S_P, S_Q : (\neg e \wedge \mathbf{R})[S_P] \wedge rsem(N,S_P,s_2,S_Q) \Rightarrow \mathbf{Q}[S_Q]) \tag{2}$$
$$\Rightarrow (\mathbf{P}[S] \wedge rsem(N,S,c,SQ) \Rightarrow \mathbf{Q}[SQ])$$

From the definition of *rssem* (see Sect. 3), we get the following semantic property:

$$rssem(N,S,\texttt{if}(e)\{s_1\}\texttt{else}\{s_2\},SQ) \Leftrightarrow (S(e) \wedge rssem(N,S,s_1,SQ)) \vee (\neg S(e) \wedge rssem(N,S,s_2,SQ)) \tag{3}$$

We derive $\mathbf{Q}[SQ]$ from $\mathbf{P}[S] \wedge rsem(N,S,c,SQ)$ using the premises of the rule and the semantic property in the derivation:

$\mathbf{P}[S] \wedge rsem(N,S,c,SQ)$

$\Longrightarrow$ [[definition of *rsem*]]
$\mathbf{P}[S] \wedge wtp(S,c) \wedge rssem(N,S,c,SQ)$
$\Longrightarrow$ [[$wtp(S,c) = wtp(S,s_1) \wedge wtp(S,s_2)$ because $c \equiv$ "$\texttt{if}(e)\{s_1\}\texttt{else}\{s_2\}$" and $vis(c) = vis(s_1) = vis(s_2)$]]
$\mathbf{P}[S] \wedge wtp(S,s_1) \wedge wtp(S,s_2) \wedge rssem(N,S,c,SQ)$
$\Longrightarrow$ [[semantical property (3)]]
$\mathbf{P}[S] \wedge wtp(S,s_1) \wedge wtp(S,s_2)$
$((S(e) \wedge rssem(N,S,s_1,SQ)) \vee (\neg S(e) \wedge rssem(N,S,s_2,SQ)))$
$\Longrightarrow$ [[predicate logic]]
$(\mathbf{P}[S] \wedge wtp(S,s_1) \wedge S(e) \wedge rssem(N,S,s_1,SQ))$
$\vee (\mathbf{P}[S] \wedge wtp(S,s_2) \wedge \neg S(e) \wedge rssem(N,S,s_2,SQ))$
$\Longrightarrow$ [[definition of *rsem*]]
$(\mathbf{P}[S] \wedge S(e) \wedge rsem(N,S,s_1,SQ)) \vee (\mathbf{P}[S] \wedge \neg S(e) \wedge rsem(N,S,s_2,SQ))$
$\Longrightarrow$ [[apply (1)]]
$\mathbf{Q}[SQ] \vee (\mathbf{P}[S] \wedge \neg S(e) \wedge rsem(N,S,s_2,SQ))$
$\Longrightarrow$ [[apply (2)]]
$\mathbf{Q}[SQ] \vee \mathbf{Q}[SQ]$
$\Longrightarrow$ [[ ]]
$\mathbf{Q}[SQ]$

QED: if-rule.

Soundness of the while-rule:
Let $c \equiv$ "$\texttt{while}(e)\{s\}$". According to lemma *quasi*, it suffices to show for all N, for all free logical variables in $\mathbf{P}$, and for all states S and SQ:

$$\begin{array}{ll} & (\ \forall S, SQ : S(e) = \texttt{true} \wedge \mathbf{P}[S] \wedge rsem(N,S,s,SQ) \Rightarrow \mathbf{P}[SQ]\ ) \\ \Rightarrow & \mathbf{P}[S] \wedge rsem(N,S,c,SQ) \Rightarrow (S(\texttt{exc}) \neq \texttt{null} \vee S(e) = \texttt{false}) \wedge \mathbf{P}[SQ] \end{array}$$

In the following proof, we use the premise in the first line above as an assumption. From that assumption, we derive the following implication (+) enusing the definition of *rsem* and the property $rssem_k(N,S,s,SQ) \Rightarrow rssem(N,S,s,SQ)$:

$$\forall S, SQ : S(e) = \texttt{true} \wedge \mathbf{P}[S] \wedge wtp(S,s) \wedge rssem_k(N,S,s,SQ) \Rightarrow \mathbf{P}[SQ]$$

By computational induction, we prove for all k that the following implication (*) holds:

$$\mathbf{P}[S] \wedge wtp(S,c) \wedge rssem_k(N,S,c,SQ) \Rightarrow (SQ(\texttt{exc}) \neq \texttt{null} \vee SQ(e) = \texttt{false}) \wedge \mathbf{P}[SQ]$$

Within the proof, we need the following semantic property that is implied by the definition of $rssem_k$ (see Sect. 3):

$$\begin{array}{ll} rssem_{k+1}(N,S,c,SQ) \Leftrightarrow & \\ & (S(e) = \texttt{false} \wedge SQ = S) \\ \vee & (S(e) = \texttt{true} \wedge \exists SR : rssem_k(N,S,s,SR) \wedge SR(\texttt{exc}) = \texttt{null} \wedge rssem_k(N,SR,c,SQ)) \\ \vee & (S(e) = \texttt{true} \wedge rssem_k(N,S,s,SQ) \wedge SQ(\texttt{exc}) \neq \texttt{null}) \end{array}$$

Induction base:
$k = 0$, that is, the premise of (*) is false.

Induction step: $k \rightarrow k+1$, with induction hypothesis (*). We derive

$$
\begin{array}{ll}
 & \mathbf{P}[S] \wedge wtp(S,c) \wedge rssem_{k+1}(N,S,c,SQ) \\
\Rightarrow & [\![\text{ semantic property }]\!] \\
 & \mathbf{P}[S] \wedge wtp(S,c) \ \wedge \ (\\
 & \quad (S(e) = \texttt{false} \wedge SQ = S) \\
 & \vee (S(e) = \texttt{true} \wedge \exists SR : rssem_k(N,S,s,SR) \wedge SR(\texttt{exc}) = \texttt{null} \wedge rssem_k(N,SR,c,SQ)) \\
 & \vee (S(e) = \texttt{true} \wedge rssem_k(N,S,s,SQ) \wedge SQ(\texttt{exc}) \neq \texttt{null}) \\
 &) \\
\Rightarrow & [\![\ wtp(S,c) \Leftrightarrow wtp(S,s), (+), \text{ well-typed-res-induct }]\!] \\
 & \quad (\mathbf{P}[S] \wedge wtp(S,c) \wedge S(e) = \texttt{false} \wedge SQ = S) \\
 & \vee (\exists SR : \ \mathbf{P}[SR] \wedge wt(SR,s) \wedge SR(\texttt{exc}) = \texttt{null} \wedge rssem_k(N,SR,c,SQ)) \\
 & \vee (\mathbf{P}[SQ] \wedge SQ(\texttt{exc}) \neq \texttt{null}) \\
\Rightarrow & [\![\text{ predicate logic; definition of } wtp]\!] \\
 & \quad (\mathbf{P}[SQ] \wedge (SQ(\texttt{exc}) \neq \texttt{null} \vee SQ(e) = \texttt{false}) \ ) \\
 & \vee (\exists SR : \ \mathbf{P}[SR] \wedge wtp(SR,s) \wedge rssem_k(N,SR,c,SQ)) \\
\Rightarrow & [\![\text{ induction hypothesis }]\!] \\
 & \quad (\mathbf{P}[SQ] \wedge (SQ(\texttt{exc}) \neq \texttt{null} \vee SQ(e) = \texttt{false}) \ ) \\
 & \vee (\exists SR : \ (SQ(\texttt{exc}) \neq \texttt{null} \vee SQ(e) = \texttt{false}) \wedge \mathbf{P}[SQ]) \\
\Rightarrow & [\![\text{ predicate logic }]\!] \\
 & \quad (\mathbf{P}[SQ] \wedge (SQ(\texttt{exc}) \neq \texttt{null} \vee SQ(e) = \texttt{false}) \ )
\end{array}
$$

QED: while-rule.

Soundness of the catch-rule:

Let $c \equiv$ "$\texttt{try}\{s_0\}\texttt{catch}(T\ e)\{s_1\}$". According to lemma *quasi*, it suffices to show for all N, for all free logical variables in $\mathbf{P}$,$\mathbf{R}$ and $\mathbf{Q}$, and for all states S and SQ:

$$
\begin{array}{ll}
 & (\forall S_P, S_Q : \mathbf{P}[S_P] \wedge rsem(N,S_P,s_0,S_Q) \\
 & \qquad \Rightarrow (((\texttt{exc} = null \vee \tau(exc) \not\preceq T) \wedge \mathbf{Q}) \vee (\mathbf{R} \wedge \texttt{exc} \neq \texttt{null} \wedge \tau(\texttt{exc}) \preceq T))[S_Q]) \qquad (1) \\
\wedge & (\forall S_P, S_Q : \mathbf{R}[e/\texttt{exc}][S_P] \ \wedge \ rsem(N,S_P,s_1,S_Q) \Rightarrow \mathbf{Q}[S_Q]) \qquad (2) \\
\Rightarrow & (\mathbf{P}[S] \ \wedge \ rsem(N,S,c,SQ) \Rightarrow \mathbf{Q}[SQ])
\end{array}
$$

From the definition of *rssem* (see Sect. 3), we get the following semantic property:

$$
\begin{array}{l}
rssem(N,S,\texttt{try}\{s_0\}\texttt{catch}(T\ e)\{s_1\},SQ) \Leftrightarrow \\
\quad \exists SR : ((rssem(N,S,s_0,SQ) \wedge SQ(\texttt{exc}) = \texttt{null}) \\
\quad \vee (rssem(N,S,s_0,SQ) \wedge SQ(\texttt{exc}) \neq \texttt{null} \wedge \tau(SQ(\texttt{exc})) \not\preceq T) \qquad (3) \\
\quad \vee (rssem(N,S,s_0,SR) \wedge SR(\texttt{exc}) \neq \texttt{null} \wedge \tau(SR(\texttt{exc})) \preceq T \\
\qquad \wedge rssem(N,SR[e := \texttt{exc}, \texttt{exc} = null], s_1, SQ)))
\end{array}
$$

Lemma 12 (Lemma.exc-subst)**.**

$$\mathbf{R}[SR] \Leftrightarrow \mathbf{R}[e/\texttt{exc}][SR[e := SR(\texttt{exc}), \texttt{exc} := \texttt{null}]]$$

Proof of Lemma.exc-subst.

$$
\begin{array}{ll}
 & \mathbf{R}[SR] \\
\Longleftrightarrow & [\![\ e \text{ does not occur in R }]\!] \\
 & \mathbf{R}[SR[e := SR(\texttt{exc})]] \\
\Longleftrightarrow & [\![\text{ rev. simplification }]\!] \\
 & \mathbf{R}[SR[e := SR(\texttt{exc}), \texttt{exc} := \texttt{null}][\texttt{exc} := SR(\texttt{exc})]] \\
\Longleftrightarrow & [\![\text{ rev. simplification }]\!] \\
 & \mathbf{R}[SR[e := SR(\texttt{exc}), \texttt{exc} := \texttt{null}][\texttt{exc} := SR[e := SR(\texttt{exc}), \texttt{exc} := \texttt{null}](e)]]
\end{array}
$$

$\Longleftrightarrow$ [[lm.substitution]]
$\mathbf{R}[e/\texttt{exc}][SR[e := SR(\texttt{exc}), \texttt{exc} := \texttt{null}]]$

□

We derive $\mathbf{Q}[SQ]$ from $\mathbf{P}[S] \wedge rsem(N, S, c, SQ)$ using the premises of the rule and the semantic property in the derivation:

$\mathbf{P}[S] \wedge rsem(N, S, c, SQ)$
$\Longrightarrow$ [[definition of $rsem$]]
$\mathbf{P}[S] \wedge wtp(S, c) \wedge rssem(N, S, c, SQ)$
$\Longrightarrow$ [[$wtp(S, c) = wtp(S, s_0)$ because $c \equiv$ "$\texttt{try}\{s_0\}\texttt{catch}(T\ e)\{s_1\}$" and $vis(c) \Leftrightarrow vis(s_0)$]]
$\mathbf{P}[S] \wedge wtp(S, s_0) \wedge rssem(N, S, c, SQ)$
$\Longrightarrow$ [[semantical property (3)]]
$\mathbf{P}[S] \wedge wtp(S, s_0) \wedge \exists SR : ((rssem(N, S, s_0, SQ) \wedge SQ(\texttt{exc}) = \texttt{null})$
$\vee\ (rssem(N, S, s_0, SQ) \wedge SQ(\texttt{exc}) \neq \texttt{null} \wedge \tau(SQ(\texttt{exc})) \not\preceq T)$
$\vee\ (rssem(N, S, s_0, SR) \wedge SR(\texttt{exc}) \neq \texttt{null} \wedge \tau(SR(\texttt{exc})) \preceq T$
$\wedge\ rssem(N, SR[e := SR(\texttt{exc}), \texttt{exc} := null], s_1, SQ)))$
$\Longrightarrow$ [[predicate logic]]
$(\mathbf{P}[S] \wedge wtp(S, s_0) \wedge rssem(N, S, s_0, SQ) \wedge SQ(\texttt{exc}) = \texttt{null})$
$\vee\ (\mathbf{P}[S] \wedge wtp(S, s_0) \wedge rssem(N, S, s_0, SQ) \wedge SQ(\texttt{exc}) \neq \texttt{null} \wedge \tau(SQ(\texttt{exc})) \not\preceq T)$
$\vee\ \exists SR : (\mathbf{P}[S] \wedge wtp(S, s_0) \wedge rssem(N, S, s_0, SR) \wedge SR(\texttt{exc}) \neq \texttt{null} \wedge \tau(SR(\texttt{exc})) \preceq T$
$\wedge\ rssem(N, SR[v := SR(\texttt{exc}), \texttt{exc} := null], s_1, SQ))$
$\Longrightarrow$ [[definition of $rsem$]]
$(\mathbf{P}[S] \wedge rsem(N, S, s_0, SQ) \wedge SQ(\texttt{exc}) = \texttt{null})$
$\vee\ (\mathbf{P}[S] \wedge rsem(N, S, s_0, SQ) \wedge SQ(\texttt{exc}) \neq \texttt{null} \wedge \tau(SQ(\texttt{exc})) \not\preceq T)$
$\vee\ \exists SR : (\mathbf{P}[S] \wedge rsem(N, S, s_0, SR) \wedge SR(\texttt{exc}) \neq \texttt{null} \wedge \tau(SR(\texttt{exc})) \preceq T$
$\wedge\ rssem(N, SR[e := SR(\texttt{exc}), \texttt{exc} := null], s_1, SQ))$
$\Longrightarrow$ [[with lemma *well-typed-rres* (7) and definition of wtr]]
$(\mathbf{P}[S] \wedge rsem(N, S, s_0, SQ) \wedge SQ[\texttt{exc}] = \texttt{null})$
$\vee\ (\mathbf{P}[S] \wedge rsem(N, S, s_0, SQ) \wedge SQ(\texttt{exc}) \neq \texttt{null} \wedge \tau(SQ(\texttt{exc})) \not\preceq T)$
$\vee\ \exists SR : (\mathbf{P}[S] \wedge rsem(N, S, s_0, SR) \wedge SR(\texttt{exc}) \neq \texttt{null} \wedge \tau(SR(\texttt{exc})) \preceq T$
$\wedge\ wt(SR, s_0) \wedge rssem(N, SR[e := SR(\texttt{exc}), \texttt{exc} := null], s_1, SQ))$
$\Longrightarrow$ [[with definition of wt and wtp and with $styp(e) = T$]]
$(\mathbf{P}[S] \wedge rsem(N, S, s_0, SQ) \wedge SQ(\texttt{exc}) = \texttt{null})$
$\vee\ (\mathbf{P}[S] \wedge rsem(N, S, s_0, SQ) \wedge SQ(\texttt{exc}) \neq \texttt{null} \wedge \tau(SQ(\texttt{exc})) \not\preceq T)$
$\vee\ \exists SR : (\mathbf{P}[S] \wedge rsem(N, S, s_0, SR) \wedge SR(\texttt{exc}) \neq \texttt{null} \wedge \tau(SR(\texttt{exc})) \preceq T$
$\wedge\ wtp(SR[e := SR(\texttt{exc}), \texttt{exc} := null], s_1) \wedge rssem(N, SR[v := \texttt{exc}, \texttt{exc} := null], s_1, SQ))$
$\Longrightarrow$ [[definition of $rsem$]]
$(\mathbf{P}[S] \wedge rsem(N, S, s_0, SQ) \wedge SQ(\texttt{exc}) = \texttt{null})$
$\vee\ (\mathbf{P}[S] \wedge rsem(N, S, s_0, SQ) \wedge SQ(\texttt{exc}) \neq \texttt{null} \wedge \tau(SQ(\texttt{exc})) \not\preceq T)$
$\vee\ \exists SR : (\mathbf{P}[S] \wedge rsem(N, S, s_0, SR) \wedge SR(\texttt{exc}) \neq \texttt{null} \wedge \tau(SR(\texttt{exc})) \preceq T$
$\wedge\ rsem(N, SR[e := SR(\texttt{exc}), \texttt{exc} := null], s_1, SQ))$

$$
\begin{array}{l}
\Longrightarrow [\![\text{ apply (1) three times }]\!] \\
\quad ((((\texttt{exc} := null \vee \tau(exc) \not\preceq T) \wedge \mathbf{Q}) \vee (\mathbf{R} \wedge \texttt{exc} \neq \texttt{null} \wedge \tau(\texttt{exc}) \preceq T))[SQ] \wedge SQ(\texttt{exc}) = \texttt{null}) \\
\quad\quad \vee ((((\texttt{exc} := null \vee \tau(exc) \not\preceq T) \wedge \mathbf{Q}) \vee (\mathbf{R} \wedge \texttt{exc} \neq \texttt{null} \wedge \tau(\texttt{exc}) \preceq T))[SQ] \\
\quad\quad\quad \wedge SQ(\texttt{exc}) \neq \texttt{null} \wedge \tau(SQ(\texttt{exc})) \not\preceq T) \\
\quad\quad \vee \exists SR : ((((SR(\texttt{exc}) = null \vee \tau(SR(exc)) \not\preceq T) \wedge \mathbf{Q}[SR]) \\
\quad\quad\quad\quad\quad \vee (\mathbf{R}[SR] \wedge SR(\texttt{exc}) \neq \texttt{null} \wedge \tau(SR(\texttt{exc})) \preceq T)) \\
\quad\quad\quad\quad \wedge SR(\texttt{exc}) \neq \texttt{null} \wedge \tau(SR(\texttt{exc})) \preceq T \\
\quad\quad\quad\quad \wedge rsem(N, SR[e := SR(\texttt{exc}), \texttt{exc} := null], s_1, SQ)) \\
\Longrightarrow [\![\text{ predicate logic }]\!] \\
\quad (SQ(\texttt{exc}) = null \wedge \mathbf{Q}[SQ]) \\
\quad\quad \vee ((SQ(\texttt{exc}) \neq \texttt{null} \wedge \tau(SQ(exc)) \not\preceq T) \wedge \mathbf{Q}[SQ]) \\
\quad\quad \vee \exists SR : (\mathbf{R}[SR] \wedge rsem(N, SR[e := SR(\texttt{exc}), \texttt{exc} := null], s_1, SQ)) \\
\Longrightarrow [\![\text{ predicate logic }]\!] \\
\quad \mathbf{Q}[SQ] \\
\quad\quad \vee \exists SR : (\mathbf{R}[SR] \wedge rsem(N, SR[e := SR(\texttt{exc}), \texttt{exc} := null], s_1, SQ)) \\
\Longrightarrow [\![\text{ use lemma 12 }]\!] \\
\quad \mathbf{Q}[SQ] \\
\quad\quad \vee \exists SR : (\mathbf{R}[e/\texttt{exc}][SR[e := SR(\texttt{exc}), \texttt{exc} := \texttt{null}]] \\
\quad\quad\quad\quad \wedge rsem(N, SR[e := SR(\texttt{exc}), \texttt{exc} := null], s_1, SQ)) \\
\Longrightarrow [\![\text{ apply (2) }]\!] \\
\quad \mathbf{Q}[SQ] \vee \exists SR : (\mathbf{Q}[SQ]) \\
\Longrightarrow [\![\text{ predicate logic }]\!] \\
\quad \mathbf{Q}[SQ]
\end{array}
$$

QED: catch-rule.

Soundness of the block-rule:

Let $c \equiv$ "$\{T\, v; s\}$". According to lemma *quasi*, it suffices to show for all N, for all free logical variables in $\mathbf{P}$ and $\mathbf{Q}$, and for all states S and SQ:

$$
\begin{array}{l}
(\forall S_P, S_Q : (\mathbf{P} \wedge v = init(T))[S_P] \wedge rsem(N, S_P, s, S_Q) \Rightarrow \mathbf{Q}[S_Q]) \\
\Rightarrow (\mathbf{P}[S] \wedge rsem(N, S, cs, SQ) \Rightarrow \mathbf{Q}[SQ])
\end{array} \tag{1}
$$

From the definition of *rssem* (see Sect. 3), we get the following semantic property:

$$
\begin{array}{l}
rssem(N, S, c, SQ) \Leftrightarrow \\
\quad rssem(N, S[v := init(T)], s, SQ)
\end{array} \tag{2}
$$

We derive $\mathbf{Q}[SQ]$ from $\mathbf{P}[S] \ \wedge \ rsem(N, S, c, SQ)$ using the premises of the rule and the semantic property in the derivation:

$$
\begin{array}{l}
\quad \mathbf{P}[S] \wedge rsem(N, S, c, SQ) \\
\Longrightarrow [\![\text{ definition of } rsem]\!] \\
\quad \mathbf{P}[S] \wedge wtp(S, c) \wedge rssem(N, S, c, SQ) \\
\Longrightarrow [\![\text{ semantical property (2) }]\!] \\
\quad \mathbf{P}[S] \wedge wtp(S, c) \wedge rssem(N, S[v := init(T)], s, SQ)
\end{array}
$$

$\Longrightarrow$ [[with definition of *wtp* and *wt*]]
$\mathbf{P}[S] \wedge wtp(S[v := init(T)], s) \wedge rssem(N, S[v := init(T)], s, SQ)$
$\Longrightarrow$ [[definition of *rsem*]]
$\mathbf{P}[S] \wedge rsem(N, S[v := init(T)], s, SQ)$
$\Longrightarrow$ [[v does not occur in $\mathbf{P}$]]
$\mathbf{P}[S[v := init(T)]] \wedge rsem(N, S[v := init(T)], s, SQ)$
$\Longrightarrow$ [[predicate logic]]
$\mathbf{P}[S[v := init(T)]] \wedge S[v := init(T)][v] = init(T) \wedge rsem(N, S[v := init(T)], s, SQ)$
$\Longrightarrow$ [[apply (2)]]
$\mathbf{Q}[SQ]$

QED: block-rule.

Soundness of inv-exc:

invoc-axiom:

$$\mathcal{A} \rhd \left\{ \begin{array}{l} y = \texttt{null} \\ \wedge\ \mathbf{P}[\$\langle \texttt{NullPExc} \rangle / \$, new(\$, \texttt{NullPExc}) / \texttt{exc}] \end{array} \right\} x = y.m(e); \{\ \mathbf{P}\ \}$$

According to lemma.rusi and the substitution lemma, it suffices to show the following formula:

$S(y) = \texttt{null} \wedge \mathbf{P}[S[\$:= \$\langle \texttt{NullPExc} \rangle, \texttt{exc} := new(\$, \texttt{NullPExc})] \wedge rsem(N, S, c, SQ) \Rightarrow \mathbf{P}[SQ]$

Semantic property (see above) leads to:

$rssem(N, S, c, SQ) \wedge S(y) = \texttt{null} \Rightarrow SQ = S[\$:= S(\$)\langle \texttt{NullPExc} \rangle, \texttt{exc} := new(S(\$), \texttt{NullPExc})]$

Thus, we can derive:

$S(y) = \texttt{null} \wedge \mathbf{P}[S[\$:= \$\langle \texttt{NullPExc} \rangle, \texttt{exc} := new(\$, \texttt{NullPExc})] \wedge rsem(N, S, c, SQ)$
$\Rightarrow$
$\mathbf{P}[S[\$:= \$\langle \texttt{NullPExc} \rangle, \texttt{exc} := new(\$, \texttt{NullPExc})]]$
$\wedge\ SQ = S[\$:= S(\$)\langle \texttt{NullPExc} \rangle, \texttt{exc} := new(S(\$), \texttt{NullPExc})]$
$\Rightarrow$
$\mathbf{P}[SQ]$

QED: inv-exc.

Soundness of the invoc-rule: Let $c \equiv$ "$x = y.m(e);$" and $vm \equiv$ "$vm(styp(y', \text{“}x = y.m(e);\text{”}), m)$". According to lemma *quasi*, it suffices to show for all N, for all free logical variables in $\mathbf{P}$, and for all states S and SQ:

$$\begin{array}{ll} & (\ \forall S, SQ : \mathbf{P}[S] \wedge rsem(N, S, vm, SQ) \Rightarrow \mathbf{Q}[SQ]\ ) \\ \Rightarrow & S(y) \neq \texttt{null} \wedge \mathbf{P}[y/\texttt{this}, e/\texttt{par}][S] \wedge rsem(N, S, c, SQ) \Rightarrow \mathbf{Q}[x/\texttt{res}][SQ] \end{array}$$

Recall from the syntactical constraints of formulas that $\mathbf{P}$ may not contain the variable *res* and that $\mathbf{Q}$ may only contain the program variables *res*, exc, and \$. In the following proof, we use the premise in the first line above as an assumption. The operational semantics yields the following property:

$$\begin{array}{ll} & rssem_{k+1}(N, S, c, SQ) \Leftrightarrow \\ & \quad (S(e) = \texttt{false} \wedge SQ = S) \\ \vee & (S(e) = \texttt{true} \wedge \exists SR : rssem_k(N, S, s, SR) \wedge SR(\texttt{exc}) = \texttt{null} \wedge rssem_k(N, SR, c, SQ)) \\ \vee & (S(e) = \texttt{true} \wedge rssem_k(N, S, s, SQ) \wedge SQ(\texttt{exc}) \neq \texttt{null}) \end{array}$$

Semantic property:

$$
\begin{array}{ll}
& rssem(N,S,c,SR) \Leftrightarrow \\
& \quad (N>0 \ \wedge \ \exists\ SQ,\ DMI : S(y) \neq \texttt{null} \wedge \tau(S(y)) \preceq styp(y,c) \wedge DMI = dm(\tau(S(y)),m) \\
& \quad \wedge\ rssem(N-1, S[\texttt{this} := S(y), \texttt{par} := S(e), \texttt{res} := init(rtyp(DMI))], body(DMI),\ SQ) \\
& \quad \wedge\ SR = S[x := SQ(\texttt{res}),\ \$:= SQ(\$),\ \texttt{exc} := SQ(\texttt{exc})]) \\
\vee & \quad (S(e) = \texttt{true} \wedge \exists SR : rssem_k(N,S,s,SR) \wedge SR(\texttt{exc}) = \texttt{null} \wedge rssem_k(N,SR,c,SQ)) \\
\vee & \quad (S(e) = \texttt{true} \wedge rssem_k(N,S,s,SQ) \wedge SQ(\texttt{exc}) \neq \texttt{null})
\end{array}
$$

from which we derive

$$
\begin{array}{l}
\quad S(y) = \texttt{null} \wedge SR = S[\$:= S(\$)\langle \texttt{NullPExc}\rangle, \texttt{exc} := new(S(\$), \texttt{NullPExc})] \\
\Rightarrow \\
\quad rssem(N,S,c,SR)
\end{array}
$$

Case distinction on the call depth N:

Case N=0:

$$
\begin{array}{ll}
& ((S(y) \neq \texttt{null} \wedge \mathbf{P}[y/this, e/par][S]) \vee \\
& (S(y) = \texttt{null} \wedge \mathbf{Q}[S[\$:= S(\$)\langle \texttt{NullPExc}\rangle, \texttt{exc} := new(S(\$), \texttt{NullPExc})]])) \\
& \wedge\ rsem(0,S,c,SQ) \\
\Rightarrow & [\![\text{ semantic property }]\!] \\
& ((S(y) \neq \texttt{null} \wedge \mathbf{P}[y/this, e/par][S]) \vee \\
& (S(y) = \texttt{null} \wedge \mathbf{Q}[S[\$:= S(\$)\langle \texttt{NullPExc}\rangle, \texttt{exc} := new(S(\$), \texttt{NullPExc})]])) \\
& \wedge\ S(y) = \texttt{null} \wedge SQ = S[\$:= S(\$)\langle \texttt{NullPExc}\rangle, \texttt{exc} := new(S(\$), \texttt{NullPExc}) \\
\Rightarrow & [\![\text{ "exclusion of Third" }]\!] \\
& S(y) = \texttt{null} \wedge \mathbf{Q}[S[\$:= S(\$)\langle \texttt{NullPExc}\rangle, \texttt{exc} := new(S(\$), \texttt{NullPExc})]] \\
& \wedge\ SQ = S[\$:= S(\$)\langle \texttt{NullPExc}\rangle, \texttt{exc} := new(S(\$), \texttt{NullPExc}) \\
\Rightarrow & [\![\text{ "insertion of } SQ \text{ }]\!] \\
& \mathbf{Q}[SQ]
\end{array}
$$

XXXXXXX Hier weiter!

Case N < 0:

It suffices to show for arbitrary N (but fixed) that under the assumptions

$$(\forall S, SQ : \mathbf{P}[S] \wedge rsem(N+1, S, vm(styp(y', c), m), SQ) \Rightarrow \mathbf{Q}[SQ])$$

and

$$S(y) \neq \texttt{null} \wedge \mathbf{P}[y/this, e/par][S] \wedge rsem(N+1, S, c, SQ)$$

we can prove

$$\mathbf{Q}[x/res][SQ]$$

Using the definition of *rsem*we get from the first assumption:

$$
\begin{array}{ll}
& (\forall S, SQ : \mathbf{P}[S] \wedge rsem(N+1, S, vm(styp(y', c), m), SQ) \Rightarrow \mathbf{Q}[SQ]) \\
\Leftrightarrow & \\
& (\forall S, SQ : \mathbf{P}[S] \wedge wtp(S, vm(styp(y', c), m)) \wedge rsem(N+1, S, dm(\tau(S(this)), m), SQ)) \\
& \Rightarrow \mathbf{Q}[SQ]) \\
\Leftrightarrow & \\
& (\forall S, SQ : \mathbf{P}[S] \wedge wtp(S, vm(styp(y', c), m)) \wedge wtp(S, dm(\tau(S(this)), m)) \wedge \\
& rsem(N, S[res := init(rtyp(dm(\tau(S(this)), m)))], body(dm(\tau(S(this)), m)), SQ)) \\
& \Rightarrow \mathbf{Q}[SQ])
\end{array}
$$

Substitution of S[this:=S(y),par:=e[S]] for S and SR for SQ yields

$\mathbf{P}[S[this := S(y), par := e[S]]] \wedge wtp(S[this := S(y), par := e[S]], vm(styp(y', c), m))$
$\wedge\ wtp(S[this := S(y), par := e[S]], dm(\tau(S(y)), m))$
$\wedge\ rsem(N, S[this := S(y), par := e[S], res := init(rtyp(dm(\tau(S(y)), m))))],$
$body(dm(\tau(S(y)), m)), SR))$
$\Rightarrow \mathbf{Q}[SR]$

$\Rightarrow$ [[def of *rsem*]]

$\mathbf{P}[S[this := S(y), par := e[S]]] \wedge wtp(S[this := S(y), par := e[S]], vm(styp(y', c), m))$
$\wedge\ wtp(S[this := S(y), par := e[S]], dm(\tau(S(y)), m))$
$\wedge\ wtp(S[this := S(y), par := e[S], res := init(rtyp(dm(\tau(S(y)), m))))], dm(\tau(S(y)), m))$
$\wedge\ rssem(N, S[this := S(y), par := e[S], res := init(rtyp(dm(\tau(S(y)), m))))],$
$body(dm(\tau(S(y)), m)), SR))$
$\Rightarrow \mathbf{Q}[SR]$

$\Rightarrow$ $[[wtp(S, c) \Rightarrow wtp(S[this := S(y), par := e[S]], vm(styp(y', c), m)) \wedge$
$wtp(S[this := S(y), par := e[S]], dm(\tau(S(y)), m)) \wedge$
$wtp(S[this := S(y), par := e[S], res := init(rtyp(dm(\tau(S(y)), m))))], dm(\tau(S(y)), m))$
note: res is not visible before method abstractions]]
$\mathbf{P}[S[this := S(y), par := e[S]]] \wedge wtp(S, c)$
$\wedge\ rssem(N, S[this := S(y), par := e[S], res := init(rtyp(dm(\tau(S(y)), m))))],$
$body(dm(\tau(S(y)), m)), SR))$

$\Rightarrow \mathbf{Q}[SR]$

We denote the last formula by (&).

We derive:

$S(y) \neq \texttt{null} \wedge \mathbf{P}[y/this, e/par][S] \wedge rsem(N+1, S, c, SQ)$

$\Rightarrow$ [[definition *rsem*]]

$S(y) \neq \texttt{null} \wedge \mathbf{P}[y/this, e/par][S] \wedge rssem(N+1, S, c, SQ) \wedge wtp(S, c)$

$\Rightarrow$ [[semp]]

$\exists SR : \mathbf{P}[y/this, e/par][S] \wedge wtp(S, c) \wedge \tau(S(y)) \preceq styp(y', c)$
$\wedge\ rssem(N, S[this := S(y), par := e[S], res := init(rtyp(dm(\tau(S(y)), m)))],$
$body(dm(\tau(S(y)), m)), SR)$
$\wedge\ SQ = S[x := SR(res), \$:= SR(\$), \texttt{exc} := SR(\texttt{exc})]$

$\Rightarrow$ [[substitution lemma, definition of wtp]]

$\exists SR : \mathbf{P}[S[this := S(y), par := e[S]]] \wedge wtp(S, c)$
$\wedge\ rssem(N, S[this := S(y), par := e[S], res := init(rtyp(dm(\tau(S(y)), m)))],$
$\wedge\ SQ = S[x := SR(res), \$:= SR(\$), \texttt{exc} := SR(\texttt{exc})])$

$\Rightarrow$ [[formula &]]

$\exists SR : \mathbf{Q}[SR] \wedge SQ = S[x := SR(res), \$:= SR(\$), \texttt{exc} := SR(\texttt{exc})]$

$\Rightarrow$ [[as **Q** only contains res, `exc`, and \$, $\mathbf{Q}[SR] \Leftrightarrow \mathbf{Q}[SQ[res := SQ(x)]]$]]

$\mathbf{Q}[SQ[res := x]]$

$\Rightarrow$ [[substitution lemma]]

$\mathbf{Q}[x/res][SQ]$

QED invoc-rule.

We need the following two lemmas:

Lemma 13 (Lemma.state-subst)**.** $R[w/X][S] == R[S][S(w)/X]$

PROOF: by induction over the term structure QED Lemma.state-subst

Lemma 14 (Lemma.invoc/call-var)**.** *If w is different from x and* `exc`*, we have*

$$rssem(N, S, c, SQ) \Rightarrow S(w) = SQ(w)$$

where $c == x = y.m(e);$ *or* $c == x = super_T.m(e);$

PROOF: Considering the different statements in turn and for each we use a case distinction on N:

1. Let c == x=y.m(e); **Case NN=0:**

$rssem(0, S, c, SQ)$
$\Rightarrow$ [[semantic property]]
$SQ = S[\$:= S(\$)\langle \texttt{NullPExc} \rangle, \texttt{exc} := new(S(\$), \texttt{NullPExc})]$
$\Rightarrow$ [[w different from exc]]
$SQ(w) = S[\$:= S(\$)\langle \texttt{NullPExc} \rangle, \texttt{exc} := new(S(\$), \texttt{NullPExc})](w) = S(w)$

Case NN>0:

For $S(y) = \texttt{null}$ the reasoning is the same as in case NN=0. Now assume $S(y) \neq \texttt{null}$, and let $N + 1 = NN$:
$rssem(N + 1, S, x = y.m(e);, SQ) \wedge S(y) \neq \texttt{null}$
$\Rightarrow$ [[semantic property]]
$\exists SR : SQ = S[x := SR(res), \$:= SR(\$), \texttt{exc} := SR(\texttt{exc})]$
$\Rightarrow$ [[w different from exc]]
$\exists SR : SQ(w) = S[x := SR(res), \$:= SR(\$), \texttt{exc} := SR(\texttt{exc})](w)$
$\Rightarrow$
$SQ(w) = S(w)$

2. Let $c == x = super_T.m(e)$; **Case NN=0:**

For $c == x = super_T.m(e);: rssem(0, S, c, SQ) \Leftrightarrow false$
thus $rssem(0, S, c, SQ) \Rightarrow S(w) = SQ(w)$

Case NN>0:

Let N+1=NN:
$rssem(N + 1, S, c, SQ)$
$\Rightarrow$ [[semantic property]]
$\exists SR : SQ = S[x := SR(res), \$:= SR(\$), \texttt{exc} := SR(\texttt{exc})]$
$\Rightarrow$ [[w different from x and exc]]
$\exists SR : SQ(w) = S[x := SR(res), \$:= SR(\$), \texttt{exc} := SR(\texttt{exc})](w)$
$\Rightarrow$
$SQ(w) = S(w)$
QED invoc/call-var
QED: inv-rule.

Soundness of the invoc-var-rule:

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 w is different from x and exc.
According to the weak lemma.rusi, it suffices to show the following formula:

$$\forall Z : (\forall S, SQ : \mathbf{P}[S] \wedge rsem(N, S, c, SQ) \Rightarrow \mathbf{Q}[SQ])$$
$\Rightarrow$
$$\mathbf{P}[w/Z][S] \wedge rsem(N, S, c, SQ) \Rightarrow \mathbf{Q}[w/Z][SQ]$$

Using the lemmas invoc/call-var and state-subst from above, we derive:

$$
\begin{array}{ll}
 & \forall Z : (\forall S, SQ : \mathbf{P}[S] \wedge rsem(N, S, c, SQ) \Rightarrow \mathbf{Q}[SQ]) \\
\Rightarrow & \text{[[PL]]} \\
 & (P[S] \wedge rsem(N, S, c, SQ) \Rightarrow \mathbf{Q}[SQ])[S(w)/Z] \\
\Rightarrow & \text{[[lemma.invoc/call-var]]} \\
 & \mathbf{P}[S][S(w)/Z] \wedge rsem(N, S, c, SQ) \Rightarrow \mathbf{Q}[SQ][S(w)/Z]) \wedge S(w) = SQ(w) \\
\Rightarrow & \text{[[lemma.state-subst, PL]]} \\
 & \mathbf{P}[w/Z][S] \wedge rsem(N, S, c, SQ) \Rightarrow \mathbf{Q}[SQ][SQ(w)/Z]) \\
\Rightarrow & \text{[[lemma.state-subst, PL]]} \\
 & \mathbf{P}[w/Z][S] \wedge rsem(N, S, c, SQ) \Rightarrow \mathbf{Q}[w/Z][SQ])
\end{array}
$$

QED: inv-var-rule.

Soundness of the call-rule:

call-rule:

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

According to lemma.rusi, it suffices to show the following formula (*):

$$
\begin{array}{ll}
 & (\forall S, SQ : \mathbf{P}[S] \wedge rsem(N, S, dm(T, m), SQ) \Rightarrow \mathbf{Q}[SQ]) \\
\Rightarrow & \\
 & \mathbf{P}[e/par][S] \wedge rsem(N, S, c, SQ) \Rightarrow \mathbf{Q}[x/res][SQ]
\end{array}
$$

Semantic property:

$$
\begin{array}{ll}
 & rssem(N + 1, S, c, SR) \\
\Leftrightarrow & \\
 & \exists SQ : rssem(N, S[par := e[S], res := init(rtyp(dm(T, m)))], body(dm(T, m)), SQ) \\
 & \wedge\ SR = S[x := SQ(res), \$:= SQ(\$), \texttt{exc} := SQ(\texttt{exc})]
\end{array}
$$

$rssem(0, S, c, SR) \Leftrightarrow false$

Case distinction on the call depth N:

Case NN=0:

According to the semantic property, for NN=0 the premise in the conclusion is false.

Case NN>0:

Let N+1=NN It suffices to show for arbitrary N (but fixed) that under the assumptions

$$(\forall S, SQ : \mathbf{P}[S] \wedge rsem(N + 1, S, dm(T, m), SQ) \Rightarrow \mathbf{Q}[SQ])$$

and

$$\mathbf{P}[e/par][S] \wedge rsem(N + 1, S, c, SQ)$$

we can prove

$$\mathbf{Q}[x/res][SQ]$$

Using the definition of *rsem*, we get from the first assumption:

$(\forall S, SQ : \mathbf{P}[S] \wedge rsem(N+1, S, dm(T,m), SQ) \Rightarrow \mathbf{Q}[SQ])$

$\Leftrightarrow$

$(\forall S, SQ : \mathbf{P}[S] \wedge wtp(S, dm(T,m))$
$\wedge\ rsem(N, S[res := init(rtyp(dm(T,m)))], body(dm(T,m)), SQ)$
$\Rightarrow \mathbf{Q}[SQ])$

$\Rightarrow$ $[[wtp(S, dm(T,m)) \Rightarrow wtp(S[res := init(rtyp(dm(T,m)))], body(dm(T,m)))]]$
$(\forall S, SQ : \mathbf{P}[S] \wedge wtp(S, dm(T,m))$
$\wedge\ rssem(N, S[res := init(rtyp(dm(T,m)))], body(dm(T,m)), SQ)$
$\Rightarrow \mathbf{Q}[SQ])$

Substitution of S[par:=e[S]] for S and SR for SQ yields

$\mathbf{P}[S[par := e[S]]] \wedge wtp(S[par := e[S]], dm(T,m))$
$\wedge\ rssem(N, S[par := e[S], res := init(rtyp(dm(T,m)))], body(dm(T,m)), SR)$
$\Rightarrow \mathbf{Q}[SR])$

$\Rightarrow$ $[[wtp(S,c) \Rightarrow wtp(S[par := e[S]], dm(T,m))]]$
$\mathbf{P}[S[par := e[S]]] \wedge wtp(S,c)$
$\wedge\ rssem(N, S[par := e[S], res := init(rtyp(dm(T,m)))], body(dm(T,m)), SR)$
$\Rightarrow \mathbf{Q}[SR])$

We denote the last formula by (&).

We derive:

$\mathbf{P}[e/par][S] \wedge rsem(N+1, S, c, SQ)$

$\Rightarrow$ [[definition *rsem*]]
$\mathbf{P}[e/par][S] \wedge wtp(S,c) \wedge rssem(N+1, S, c, SQ)$

$\Rightarrow$ [[semantic property, substitution lemma]]
$\exists SR : \mathbf{P}[S[par := e[S]]] \wedge wtp(S,c) \wedge$
$rssem(N, S[par := e[S], res := init(rtyp(dm(T,m)))], body(dm(T,m)), SR)$
$\wedge\ SQ = S[x := SR(res), \$:= SR(\$), \texttt{exc} := SR(\texttt{exc})])$

$\Rightarrow$ [[formula &]]
$\exists SR : \mathbf{Q}[SR] \wedge SQ = S[x := SR(res), \$:= SR(\$), \texttt{exc} := SR(\texttt{exc})]$

$\Rightarrow$ [[as $\mathbf{Q}$ only contains res, `exc`, and \$, $\mathbf{Q}[SR] \Leftrightarrow \mathbf{Q}[SQ[res := SQ(x)]]$]]
$\mathbf{Q}[SQ[res := x]]$

$\Rightarrow$ [[substitution lemma]]
$\mathbf{Q}[x/res][SQ]$

QED: call-rule.

Soundness of the call-var-rule:

call-var-rule:

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

where w is different from x and `exc`.

Proof is textual identical to the proof of invoc-var-rule.

QED: call-var-rule.

Soundness of the impl-rule:

impl-rule:

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

According to the semantics for sequents, we may assume the antecedent

$$(\forall N : (SEM(A,N) \wedge UC(\forall S, SQ : \mathbf{P}[S] \wedge rsem(N,S,T@m,SQ) \Rightarrow \mathbf{Q}[SQ])$$
$$\Rightarrow$$
$$UC(\forall S, SQ : \mathbf{P}[S] \wedge S(res) = init(rtyp(T@m)) \wedge rsem(N,S,body(T@m),SQ) \Rightarrow \mathbf{Q}[SQ])))$$

to show

$$\forall N : SEM(A,N) \Rightarrow UC(\forall S, SQ : \mathbf{P}[S] \wedge rsem(N,S,T@m,SQ) \Rightarrow \mathbf{Q}[SQ])$$

We show this by induction on N:

Induction base:

$rsem(0,S,T@m,SQ) \Leftrightarrow false$, i.e. the goal is valid.

Induction step: $N \Rightarrow N+1$

Assuming the induction hypo:

$$SEM(A,N) \Rightarrow UC(\forall S, SQ : \mathbf{P}[S] \wedge rsem(N,S,T@m,SQ) \Rightarrow \mathbf{Q}[SQ])$$

we have to show:

$$SEM(A,N+1) \Rightarrow UC(\forall S, SQ : \mathbf{P}[S] \wedge rsem(N+1,S,T@m,SQ) \Rightarrow \mathbf{Q}[SQ])$$

Modus ponens applied to antecedent and ind. hypo yields for every S, SQ:

$\mathbf{P}[S] \wedge S(res) = init(rtyp(T@m)) \wedge rsem(N,S,body(T@m),SQ) \Rightarrow \mathbf{Q}[SQ]$
$\Rightarrow$ [[subst. S by $S[res := init(rtyp(T@m))$]]
$\mathbf{P}[S[res := init(rtyp(T@m))]] \wedge init(rtyp(T@m)) = init(rtyp(T@m))$
$\wedge\, rsem(N, S[res := init(rtyp(T@m))], body(T@m), SQ)$
$\Rightarrow \mathbf{Q}[SQ]$
$\Rightarrow$ [[$\mathbf{P}$ only depends on this, par, \$; strenthening]]
$\mathbf{P}[S] \wedge wtp(S,T@m) \wedge rsem(N, S[res := init(rtyp(T@m))], body(T@m), SQ)$
$\Rightarrow \mathbf{Q}[SQ]$
$\Rightarrow$ [[rsemd, ...]]
$\mathbf{P}[S] \wedge rsem(N+1,S,T@m,SQ) \Rightarrow \mathbf{Q}[SQ]$

QED: impl-rule.

Soundness of the class-rule:

class-rule:

$$\frac{\mathcal{A} \rhd \ \{\ \tau(this) = T \wedge \mathbf{P}\ \}\, dm(T,m)\, \{\ \mathbf{Q}\ \} \qquad \mathcal{A} \rhd \ \{\ \tau(this) \prec T \wedge \mathbf{P}\ \}\, T : m\, \{\ \mathbf{Q}\ \}}{\mathcal{A} \rhd \ \{\ \mathbf{P}\ \}\, T : m\, \{\ \mathbf{Q}\ \}}$$

According to lemma.rusi, it suffices to show the following formula (*):

$$(\forall S, SQ : \tau(S(this)) = T \wedge \mathbf{P}[S] \wedge rsem(N,S,dm(T,m),SQ) \Rightarrow \mathbf{Q}[SQ])$$
$$\wedge\, (\forall S, SQ : \tau(S(this)) \prec T \wedge \mathbf{P}[S] \wedge rsem(N,S,T:m,SQ) \Rightarrow \mathbf{Q}[SQ])$$
$$\Rightarrow$$
$$\mathbf{P}[S] \wedge rsem(N,S,T:m,SQ) \Rightarrow \mathbf{Q}[SQ]$$

We derive:

$\mathbf{P}[S] \wedge rsem(N, S, T : m, SQ)$
$\Rightarrow$ [[rsemd]]
$\mathbf{P}[S] \wedge wtp(S, T : m) \wedge rsem(N, S, T : m, SQ)$
$\Rightarrow$ [[def of wtp]]
$(\tau(S(this)) = T \vee \tau(S(this)) \prec T) \wedge \mathbf{P}[S] \wedge rsem(N, S, T : m, SQ)$
$\Rightarrow$ [[PL, rsemd]]
$(\tau(S(this)) = T \wedge \mathbf{P}[S] \wedge rsem(N, S, dm(\tau(S(this)), m), SQ)) \vee$
$(\tau(S(this)) \prec T \wedge \mathbf{P}[S] \wedge rsem(N, S, T : m, SQ))$
$\Rightarrow$ [[PL]]
$(\tau(S(this)) = T \wedge \mathbf{P}[S] \wedge rsem(N, S, dm(T, m), SQ)) \vee$
$(\tau(S(this)) \prec T \wedge \mathbf{P}[S] \wedge rsem(N, S, T : m, SQ))$
$\Rightarrow$ [[premise of (*)]]
$\mathbf{Q}[SQ]$

QED: class-rule.

Soundness of the subtype-rule:

subtype-rule:

$$\frac{S \preceq T \qquad \mathcal{A} \vdash \{\ \mathbf{P}\ \}\, S : m\, \{\ \mathbf{Q}\ \}}{\mathcal{A} \vdash \{\ \tau(this) \preceq S \wedge \mathbf{P}\ \}\, T : m\, \{\ \mathbf{Q}\ \}}$$

According to lemma.rusi, it suffices to show the following formula (*):

$$(\forall S, SQ : \mathbf{P}[S] \wedge rsem(N, S, T0 : m, SQ) \Rightarrow \mathbf{Q}[SQ])$$
$$\Rightarrow$$
$$(\mathbf{P}[S] \wedge \tau(S(this)) \preceq T0 \wedge rsem(N, S, T : m, SQ) \Rightarrow \mathbf{Q}[SQ])$$

Thus, it suffices to show:

$$\tau(S(this)) \preceq T0 \wedge rsem(N, S, T : m, SQ) \Rightarrow rsem(N, S, T0 : m, SQ)$$

We derive:
$\tau(S(this)) \preceq T0 \wedge rsem(N, S, T : m, SQ)$
$\Rightarrow$ [[def of *rsem*]]
$\tau(S(this)) \preceq T0 \wedge wtp(S, T : m) \wedge rsem(N, S, dm(\tau(S(this)), m), SQ)$
$\Rightarrow$ [[def of wtp, context conditions for par of m]]
$wtp(S, T0 : m) \wedge rsem(N, S, dm(\tau(S(this)), m), SQ)$
$\Rightarrow$ [[def of *rsem*]]
$rsem(N, S, T0 : m, SQ)$

QED: subtype-rule.

Soundness of the assumpt-axiom:

assumpt-axiom:

$$\frac{}{\mathcal{A} \vdash \mathcal{A}}$$

We have to show for every triple A:

$$\forall N : A(N) \Rightarrow A(N)$$

which is valid.

QED: assumpt-axiom.

Soundness of the false-axiom:

false-axiom:

$$\frac{}{\mathcal{A} \mid\!\rhd \ \{\ false\ \}\, pp\, \{\ false\ \}}$$

We have to show

$$\forall N : false \wedge rsem(N, S, pp, SQ) \Rightarrow false$$

which is valid.

QED: false-axiom.

Soundness of the assumpt-intro-rule:

assumpt-intro-rule:

$$\frac{\mathcal{A} \mid\!\rhd a}{\mathcal{A}, a0 \mid\!\rhd a}$$

We have to show

$$\begin{array}{l} \quad (\forall N : SEM(A, N) \Rightarrow SEM(a, N)) \\ \Rightarrow \\ \quad (\forall N : SEM(A, N) \wedge SEM(a0, N) \Rightarrow SEM(a, N)) \end{array}$$

which is valid.
QED: assumpt-intro-rule.

Soundness of the assumpt-elim-rule:

assumpt-elim-rule:

$$\frac{\mathcal{A} \mid\!\rhd a0 \quad \mathcal{A}, a0 \mid\!\rhd a}{\mathcal{A} \mid\!\rhd a}$$

We have to show

$$\begin{array}{l} \quad (\forall N : SEM(A, N) \Rightarrow SEM(a0, N)) \\ \quad \wedge\, (\forall N : SEM(A, N) \wedge SEM(a0, N) \Rightarrow SEM(a, N)) \\ \Rightarrow \\ \quad (\forall N : SEM(A, N) \Rightarrow SEM(a, N)) \end{array}$$

which is valid.
QED: assumpt-elim-rule.

Soundness of the disjunct-rule:

disjunct-rule:

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

According to lemma.rusi, it suffices to show the following formula (*):

$$\begin{aligned} & (\forall S, SQ : \mathbf{P}1[S] \wedge rsem(N, S, pp, SQ) \Rightarrow \mathbf{Q}1[SQ]) \\ & \wedge (\forall S, SQ : \mathbf{P}2[S] \wedge rsem(N, S, pp, SQ) \Rightarrow \mathbf{Q}2[SQ]) \\ \Rightarrow & \\ & (P1[S] \vee \mathbf{P}2[s]) \wedge rsem(N, S, pp, SQ) \Rightarrow \mathbf{Q}1[SQ] \vee \mathbf{Q}2[SQ] \end{aligned}$$

which is valid.
QED: disjunct-rule.

Soundness of the ex-rule:

ex-rule:

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

According to lemma.rusi, it suffices to show the following formula (*):

$$\begin{aligned} & (\forall S, SQ : \mathbf{P}[S] \wedge rsem(N, S, pp, SQ) \Rightarrow \mathbf{Q}[Y/Z][SQ]) \\ \Rightarrow & \\ & (\exists Z : \mathbf{P}[S] \wedge rsem(N, S, pp, SQ)) \Rightarrow \mathbf{Q}[Y/Z][SQ] \end{aligned}$$

which is valid because Z does not occur free in $\mathbf{Q}[Y/Z]$.
QED: ex-rule.

Soundness of the swis-rule:

swis-rule:

$$\frac{\begin{array}{l} PVSUBSTBYLVAR(\mathbf{PP} \wedge \tau(v) \preceq styp(v', pp) \wedge this \neq \texttt{null} \wedge \texttt{exc} = \texttt{null} \Rightarrow \mathbf{P}) \\ \mathcal{A} \vdash \{\ \mathbf{P}\ \}\, pp\, \{\ \mathbf{Q}\ \} \\ PVSUBSTBYLVAR(\mathbf{Q} \wedge \tau(v) \preceq styp(v', pp) \wedge this \neq \texttt{null} \Rightarrow \mathbf{QQ}) \end{array}}{\mathcal{A} \vdash \{\ \mathbf{PP}[t/Z] \wedge R\ \}\, pp\, \{\ \mathbf{QQ}[t/Z] \wedge R\ \}}$$

where v is a program variable of pp, Z is a logical variable, t is a Sigma-term, and R is a Sigma-formula not depending on program variables, and **PP** and **QQ** are admissable pre-/postformulas for pp.

Based on the assumptions

$$\begin{aligned} \mathbf{PP} \wedge \tau(v) \preceq styp(v', pp) \wedge this \neq \texttt{null} \wedge \texttt{exc} = \texttt{null} &\Rightarrow \mathbf{P} \\ \forall S, SQ : \mathbf{P}[S] \wedge rsem(N, S, pp, SQ) &\Rightarrow \mathbf{Q}[SQ] \\ \mathbf{Q} \wedge \tau(v) \preceq styp(v', pp) \wedge this \neq \texttt{null} &\Rightarrow \mathbf{QQ} \end{aligned}$$

we have to show

$$\mathbf{PP}[t/Z][S] \wedge R[S] \wedge rsem(N, S, pp, SQ) \Rightarrow \mathbf{QQ}[t/Z][SQ] \wedge R[SQ]$$

As $R[S] == R[SQ]$ and $\mathbf{PP}[t/Z][S] == \mathbf{PP}[S][t/Z]$, $\mathbf{QQ}[t/Z][SQ] == \mathbf{QQ}[SQ][t/Z]$ it suffices to show:

$$\mathbf{PP}[S] \wedge rsem(N, S, pp, SQ) \Rightarrow \mathbf{QQ}[SQ]$$

We derive:

$\mathbf{PP}[S] \wedge rsem(N, S, pp, SQ)$
$\Rightarrow$ [[def $rsem$]]
$\mathbf{PP}[S] \wedge wtp(S, pp) \wedge rsem(N, S, pp, SQ)$
$\Rightarrow$ $[[defofwtp]]$
$\mathbf{PP}[S] \wedge \tau(S(v) \preceq styp(v', pp) \wedge S(this) \neq \texttt{null}$
$\wedge\ S(\texttt{exc}) = \texttt{null} \wedge rsem(N, S, pp, SQ)$
$\Rightarrow$ [[assumption]]
$\mathbf{P}[S] \wedge rsem(N, S, pp, SQ)$
$\Rightarrow$ $[[well - typed - nres]]$
$\mathbf{P}[S] \wedge rsem(N, S, pp, SQ) \wedge wtr(SQ, pp)$
$\Rightarrow$ [[assumption]]
$\mathbf{Q}[SQ] \wedge wtr(SQ, pp)$
$\Rightarrow$ [[def of wtr]]
$\mathbf{Q}[SQ] \wedge \tau(S(v) \preceq styp(v', pp) \wedge S(this) \neq \texttt{null}$
$\Rightarrow$ [[assumption]]
$\mathbf{QQ}[SQ]$

QED: swiss-rule.

6 Completeness proof

It turns out that following axioms and rules are not needed for completeness:

- true-axiom
- conjunct-rule
- all-rule

We use the following notation:

- ASSUMPT $\rhd$ TRIPLE denotes a sequent; `if` the premise is empty the seperation sign can be dropped.
- $\vdash$ SEQ states that there is a formal proof for SEQ in the programming logic.
- $\models$ SEQ states that SEQ is valid in the SV-framework (note that SEQ is a syntactical abbreviation).

Let M be a given package. Syntactical abbreviations:

$$\begin{array}{rcll} \bigwedge_{v\in vis(pp)} F(v) & == & F(v1) \ \wedge \ \ldots \ \wedge \ F(vn) & \texttt{if } vis(pp) = \{v1,..,vn\} \\ \pi(S,pp) & == & \bigwedge_{v\in vis(pp)} S(v) = v \ \wedge \ S(\$) = \$ & \\ \rho(S,m) & == & S(res) = res \ \wedge \ S(exc) = exc \ \wedge \ S(\$) = \$ & \\ \rho(S,c) & == & \pi(S,c) & \end{array}$$

We have to show:

If $\models UC(\forall S, SQ : \mathbf{P}[S] \ \wedge \ sem(S,pp,SQ) \Rightarrow Q[SQ])$

then $\vdash \{\mathbf{P}\}pp\{\mathbf{Q}\}$

where $\mathbf{P}, \mathbf{Q}$ do not contain a free S, SQ.

In the following proof annotations, we write

inva for the invariant part of swis

strg for strengthening

weak for weakening

semp for the given semantic properties of the language

semd for the extension of *sem* to method abstractions.

In a first step, we use the programming logic to show for all pp of M the most general formula MGF:

$$\vdash \{\pi(S,pp)\}pp\{\exists SQ : sem(S,pp,SQ) \ \wedge \ \rho(SQ,pp)\}$$

From MFG we can derive the desired as follows:

$$\frac{\left\{ \ \pi(S,pp) \ \right\} pp \left\{ \ \exists SQ : sem(S,pp,SQ) \ \wedge \ \rho(SQ,pp) \ \right\}}{\left\{ \ \mathbf{P}[S] \ \wedge \ \pi(S,pp) \ \right\} pp \left\{ \ \mathbf{P}[S] \ \wedge \ \exists SQ : sem(S,pp,SQ) \ \wedge \ \rho(SQ,pp) \ \right\}} \text{[inva]}$$

$$\frac{\left\{ \ \mathbf{P}[S] \ \wedge \ \pi(S,pp) \ \right\} pp \left\{ \ \exists SQ : \mathbf{P}[S] \ \wedge \ sem(S,pp,SQ) \ \wedge \ \rho(SQ,pp) \ \right\}}{\left\{ \ \pi(S,pp) \ \wedge \ \mathbf{P}[S] \ \right\} pp \left\{ \ \exists SQ : \mathbf{Q}[SQ] \ \wedge \ \rho(SQ,pp) \ \right\}} \text{[swis]}$$

$$\frac{\left\{ \ \pi(S,pp) \ \wedge \ \mathbf{P}[S] \ \right\} pp \left\{ \ \exists SQ : \mathbf{Q}[SQ] \ \wedge \ \rho(SQ,pp) \ \right\}}{\left\{ \ \pi(S,pp) \ \wedge \ \mathbf{P}[S] \ \right\} pp \left\{ \ \mathbf{Q} \ \right\}} \text{[weak,semp]}$$

$$\frac{\left\{ \ \pi(S,pp) \ \wedge \ \mathbf{P}[S] \ \right\} pp \left\{ \ \mathbf{Q} \ \right\}}{\left\{ \ \exists S : \pi(S,pp) \ \wedge \ \mathbf{P}[S] \ \right\} pp \left\{ \ \mathbf{Q} \ \right\}} \text{[weak]}$$

[ex]

$$\frac{}{\{\ \mathbf{P}\ \}\, pp\, \{\ \mathbf{Q}\ \}}\ [\text{strg}]$$

It remains to show MGF for all *pp*!

Proof of MGF:

Let $mgf(pp)$ abbreviate $\{\ \pi(S,pp)\ \}\, pp\, \{\ \exists SQ : sem(S,pp,SQ)\ \wedge\ \rho(SQ,pp)\ \}$. Let $MV = \{mv^1,\ldots,mv^q\}$ be the set of virtual methods and $MI = \{mi^1,\ldots,mi^r\}$ be the set of implemented methods of M.

$$\begin{array}{rcl} \mathcal{AV} & = & \{mgf(mv^i) | mv^i \ in\ MV\} \\ \mathcal{AI} & = & \{mgf(mi^j) | mi^j \ in\ MI\} \\ \mathcal{AVI} & = & \mathcal{AV} \cup \mathcal{AI} \end{array}$$

Proof structure:

1. Derive $\mathcal{AVI} \mid\!\!\rhd\ mgf(c)$ for all statements c by induction on the depth of c's syntax tree.
2. Derive $\mathcal{AI} \mid\!\!\rhd\ mgf(mv^i)$ for all mv^i by induction on the type hierarchy.
3. Derive $\mathcal{AI} \mid\!\!\rhd\ mgf(c)$ for all statements by assumption elimination from (1)+(2)
4. Derive $mgf(mi^{j+1}),\ldots,mgf(mi^r) \mid\!\!\rhd\ mgf(mi^j)$ for all mi^j
   from $mgf(mi^j),\ldots,mgf(mi^r) \mid\!\!\rhd\ mgf(body(mi^j))$ based on the results of (3)
5. Derive $\mid\!\!\rhd\ mgf(mi^j)$ for all mi^j by assumption elimination.
6. Derive $\mid\!\!\rhd\ mgf(mv^i)$ for all mv^i by assumption elimination.
7. Derive $\mid\!\!\rhd\ mgf(c)$ for all statements by assumption elimination.

AD 1:

To derive $\mathcal{AVI} \mid\!\!\rhd\ mgf(c)$ for all statements, we use induction over the structure of c and exploit the properties of *sem*. In some steps, we show only mgf_t and mgf_d

$$\begin{array}{lcl} \mathcal{AVI} & \mid\!\!\rhd & \{\ \pi(S,c)\ \wedge\ sem(S,c,SQ)\ \}\, c\, \{\ \pi(SQ,c)\ \} \\ \mathcal{AVI} & \mid\!\!\rhd & \{\ \pi(S,c)\ \wedge\ \neg\exists SQ : sem(S,c,SQ)\ \}\, c\, \{\ false\ \} \end{array}$$

From that, we can derive *mgf* as follows:

$$\frac{\mathcal{AVI} \mid\!\!\rhd\ \{\ \pi(S,c)\ \wedge\ sem(S,c,SQ)\ \}\, c\, \{\ \pi(SQ,c)\ \}}{\mathcal{AVI} \mid\!\!\rhd\ \{\ \pi(S,c)\ \wedge\ sem(S,c,SQ)\ \}\, c\, \{\ sem(S,c,SQ)\ \wedge\ \pi(SQ,c)\ \}}\ [\text{inva}]$$

$$\frac{\mathcal{AVI} \mid\!\!\rhd\ \{\ \pi(S,c)\ \wedge\ sem(S,c,SQ)\ \}\, c\, \{\ sem(S,c,SQ)\ \wedge\ \pi(SQ,c)\ \}}{\mathcal{AVI} \mid\!\!\rhd\ \{\ \pi(S,c)\ \wedge\ sem(S,c,SQ)\ \}\, c\, \left\{\ \begin{array}{l} \exists SQ : sem(S,c,SQ) \\ \wedge\ \pi(SQ,c) \end{array}\ \right\}}\ [\text{weak}]$$

$$\frac{\mathcal{AVI} \mid\!\!\rhd\ \{\ \pi(S,c)\ \wedge\ sem(S,c,SQ)\ \}\, c\, \left\{\ \begin{array}{l} \exists SQ : sem(S,c,SQ) \\ \wedge\ \pi(SQ,c) \end{array}\ \right\}}{\mathcal{AVI} \mid\!\!\rhd\ \left\{\ \begin{array}{l} \pi(S,c) \\ \wedge\ \exists SQ : sem(S,c,SQ) \end{array}\ \right\}\, c\, \left\{\ \begin{array}{l} \exists SQ : sem(S,c,SQ) \\ \wedge\ \pi(SQ,c) \end{array}\ \right\}}\ [\text{ex}]$$

Then disjuntion with

$$\mathcal{AVI} \mid\!\!\rhd\ \{\pi(S,c)\ \wedge\ \neg\exists SQ : sem(S,c,SQ)\} c \{\exists SQ : sem(S,c,SQ)\ \wedge\ \pi(SQ,c)\}$$

yields *mgf*.
On the other hand we can derive mgf_t and mgf_d from *mgf*:

$$\frac{\mathcal{AVI} \models \{\ \pi(S,c)\ \}\, c\, \{\ \exists SQ : sem(S,c,SQ)\ \wedge\ \pi(SQ,c)\ \}}{\mathcal{AVI} \models \left\{\begin{array}{l}\pi(S,c)\\ \wedge\ sem(S,c,SQ)\end{array}\right\} c \left\{\begin{array}{l}sem(S,c,SQ)\\ \wedge\ \exists SR : sem(S,c,SR)\\ \wedge\ \pi(SR,c)\end{array}\right\}}\ [\text{inva}]$$

$$\frac{\mathcal{AVI} \models \left\{\begin{array}{l}\pi(S,c)\\ \wedge\ sem(S,c,SQ)\end{array}\right\} c \left\{\begin{array}{l}sem(S,c,SQ)\\ \wedge\ \exists SR : sem(S,c,SR)\\ \wedge\ \pi(SR,c)\end{array}\right\}}\ [\text{lemma.deterministic}]$$

$$\frac{\mathcal{AVI} \models \{\ \pi(S,c)\ \wedge\ sem(S,c,SQ)\ \}\, c\, \{\ \exists SR : SQ = SR\ \wedge\ \pi(SR,c)\ \}}{\mathcal{AVI} \models \{\ \pi(S,c)\ \wedge\ sem(S,c,SQ)\ \}\, c\, \{\ \pi(SQ,c)\ \}}\ [\text{weak}]$$

$$\frac{\begin{array}{l}\mathcal{AVI} \models \{\ \pi(S,c)\ \wedge\ sem(S,c,SQ)\ \}\, c\, \{\ \pi(SQ,c)\ \}\\ \mathcal{AVI} \models \{\ \pi(S,c)\ \}\, c\, \{\ \exists SQ : sem(S,c,SQ)\ \wedge\ \pi(SQ,c)\ \}\end{array}}{\mathcal{AVI} \models \left\{\begin{array}{l}\pi(S,c)\\ \wedge\ \neg\exists SQ : sem(S,c,SQ)\end{array}\right\} c \left\{\begin{array}{l}NOT(\exists SQ : sem(S,c,SQ))\\ \wedge\ \exists SR : sem(S,c,SR)\end{array}\right\}}\ [\text{inva,weak}]$$

$$\frac{\mathcal{AVI} \models \left\{\begin{array}{l}\pi(S,c)\\ \wedge\ \neg\exists SQ : sem(S,c,SQ)\end{array}\right\} c \left\{\begin{array}{l}NOT(\exists SQ : sem(S,c,SQ))\\ \wedge\ \exists SR : sem(S,c,SR)\end{array}\right\}}\ [\text{weak}]$$

$$\mathcal{AVI} \models \{\ \pi(S,c)\ \wedge\ \neg\exists SQ : sem(S,c,SQ)\ \}\, c\, \{\ false\ \}$$

Lemma 15 (wtp (used))**.**

$$\frac{\{\ \mathbf{P}\ \wedge\ \pi(S,pp)\ \wedge\ wtp(S,pp)\ \}\, pp\, \{\ \mathbf{Q}\ \}}{\{\ \mathbf{P}\ \wedge\ \pi(S,pp)\ \}\, pp\, \{\ \mathbf{Q}\ \}}$$

PROOF: Let $v1, \ldots, vn$ be the visible variables before pp; for brevity we omit $\mathbf{P}$ in the precondition of the proof:

$$\frac{\{\ \pi(S,pp)\ \wedge\ wtp(S,pp)\ \}\, pp\, \{\ \mathbf{Q}\ \}}{\left\{\begin{array}{l}\pi(S,pp)\ \wedge\ S(\texttt{this}) \neq \texttt{null}\ \wedge\ wts(S(\$))\\ \wedge\ \tau(S(v1')) \preceq styp(v1',pp)\ \wedge\ \ldots\ \wedge\ \tau(S(vn')) \preceq styp(vn',pp)\\ \wedge\ S(\texttt{exc}) = \texttt{null}\end{array}\right\} pp\, \{\ \mathbf{Q}\ \}}\ [\text{strg}]$$

$$\frac{\left\{\begin{array}{l}\pi(S,pp)\ \wedge\ S(\texttt{this}) \neq \texttt{null}\ \wedge\ wts(S(\$))\\ \wedge\ \tau(S(v1')) \preceq styp(v1',pp)\ \wedge\ \ldots\ \wedge\ \tau(S(vn')) \preceq styp(vn',pp)\\ \wedge\ S(\texttt{exc}) = \texttt{null}\end{array}\right\} pp\, \{\ \mathbf{Q}\ \}}{\left\{\begin{array}{l}\pi(S,pp)\ \wedge\ \texttt{this} \neq \texttt{null}\ \wedge\ wts(\$)\\ \wedge\ \tau(v1) \preceq styp(v1',pp)\ \wedge\ \ldots\ \wedge\ \tau(vn) \preceq styp(vn',pp)\\ \wedge\ \texttt{exc} = \texttt{null}\end{array}\right\} pp\, \{\ \mathbf{Q}\ \}}\ [\text{strg}]$$

$$\frac{\left\{\begin{array}{l}\pi(S,pp)\ \wedge\ \texttt{this} \neq \texttt{null}\ \wedge\ wts(\$)\\ \wedge\ \tau(v1) \preceq styp(v1',pp)\ \wedge\ \ldots\ \wedge\ \tau(vn) \preceq styp(vn',pp)\\ \wedge\ \texttt{exc} = \texttt{null}\end{array}\right\} pp\, \{\ \mathbf{Q}\ \}}{\left\{\begin{array}{l}\pi(S,pp)\\ \wedge\ \tau(v1) \preceq styp(v1',pp)\ \wedge\ \ldots\ \wedge\ \tau(vn) \preceq styp(vn',pp)\end{array}\right\} pp\, \{\ \mathbf{Q}\ \}}\ [\text{swis}]$$

$$\frac{\left\{\begin{array}{l}\pi(S,pp)\\ \wedge\ \tau(v1) \preceq styp(v1',pp)\ \wedge\ \ldots\ \wedge\ \tau(vn) \preceq styp(vn',pp)\end{array}\right\} pp\, \{\ \mathbf{Q}\ \}}{\left\{\begin{array}{l}\pi(S,pp)\\ \wedge\ \tau(v1) \preceq styp(v1',pp)\ \wedge\ \ldots\ \wedge\ \tau(vn-1) \preceq styp(vn-1',pp)\end{array}\right\} pp\, \{\ \mathbf{Q}\ \}}\ [\text{swis}]$$

$$\frac{\left\{\begin{array}{l}\pi(S,pp)\\ \wedge\ \tau(v1) \preceq styp(v1',pp)\ \wedge\ \ldots\ \wedge\ \tau(vn-1) \preceq styp(vn-1',pp)\end{array}\right\} pp\, \{\ \mathbf{Q}\ \}}{\ldots\ldots [[\ n-2 \text{ steps }]]}\ [\text{swis}]$$

$$\frac{\ldots\ldots [[\ n-2 \text{ steps }]]}{\{\ \pi(S,pp)\ \wedge\ \tau(v) \preceq styp(v',pp)\ \}\, pp\, \{\ \mathbf{Q}\ \}}\ [\text{swis}]$$

$$\frac{\{\ \pi(S,pp)\ \wedge\ \tau(v) \preceq styp(v',pp)\ \}\, pp\, \{\ \mathbf{Q}\ \}}{\{\ \pi(S,pp)\ \}\, pp\, \{\ \mathbf{Q}\ \}}\ [\text{swis}]$$

QED *wtp*

Lemma 16 (ev (used))**.**

$$\pi(S,c) \Rightarrow e = e[S]$$

PROOF
NQED

In the following, we build our semantics on $ssem(!)$, not on $nsem$:

fread(VarId VarId FieldId): $c == x = y.a;$

To show:

$$\mathcal{AVI} \quad \triangleright \quad \{\ \pi(S, x = y.a;)\ \}\, x = y.a; \{\ \exists SQ : sem(S, x = y.a;, SQ)\ \wedge\ \pi(SQ, x = y.a;)\ \}$$

We show: a)

$$\mathcal{AVI} \quad \triangleright \quad \left\{ \begin{array}{l} S(y) \neq \texttt{null} \\ \wedge\ \pi(S, x = y.a;)\ \wedge\ wtp(S, c) \end{array} \right\} x = y.a; \left\{ \begin{array}{l} \exists SQ : sem(S, x = y.a;, SQ) \\ \wedge\ \pi(SQ, x = y.a;) \end{array} \right\}$$

and b)

$$\mathcal{AVI} \quad \triangleright \quad \left\{ \begin{array}{l} S(y) = \texttt{null} \\ \wedge\ \pi(S, x = y.a;)\ \wedge\ wtp(S, c) \end{array} \right\} x = y.a; \left\{ \begin{array}{l} \exists SQ : sem(S, x = y.a;, SQ) \\ \wedge\ \pi(SQ, x = y.a;) \end{array} \right\}$$

Then the disjunct-rule and the $lm.wtp$ allows to derive q.e.d.

AD a:

The property of $ssem$:

$$S(y) \neq \texttt{null} \Rightarrow \exists SQ : ssem(S, c, SQ)\ \wedge\ SQ = S[x := S(\$)(S(y).a)]$$

Using field-read-axiom on the postcondition of *mfg*, we get the following proof outline:

$\{S(y) \neq \texttt{null}\ \wedge\ \pi(S, c)\ \wedge\ wtp(S, c)\}$
$\Rightarrow$ $[[defofssemforc]]$
$\{S(y) \neq \texttt{null}\ \wedge\ \pi(S, c)\ \wedge\ wtp(S, c)\ \wedge\ \exists SQ : ssem(S, c, SQ)\ \wedge\ SQ = S[x := S(\$)(S(y).a)]\}$
$\Rightarrow$
$\{S(y) \neq \texttt{null}\ \wedge\ \pi(S, c)\ \wedge\ wtp(S, c)\ \wedge\ \exists SQ : ssem(S, c, SQ)\ \wedge\ \bigwedge_{vinvis(c)\{x\}} SQ(v) = S(v)$
$\wedge\ SQ(x) = S(\$)(S(y).a)\ \wedge\ SQ(\$) = S(\$)\}$
$\Rightarrow$ $[[strng]]$

$$\left\{ \begin{array}{l} y \neq \texttt{null}\ \wedge\ \exists SQ : sem(S, c, SQ) \\ \wedge\ \bigwedge_{vinvis(c)\{x\}} SQ(v) = v \\ \wedge\ SQ(x) = \$(S(y).a)\ \wedge\ SQ(\$) = \$ \end{array} \right\} x = y.a; \left\{ \begin{array}{l} \exists SQ : sem(S, x = y.a;, SQ) \\ \wedge\ \pi(SQ, c) \end{array} \right\}$$

AD b:

The property of $ssem$:

$$\begin{array}{rcl} S(y) & = & \texttt{null} \Rightarrow \exists SQ : ssem(S, c, SQ) \\ & & \quad \wedge\ SQ = S[\$:= S(\$)\langle \texttt{NullPExc} \rangle, \texttt{exc} := new(S(\$), \texttt{NullPExc})] \end{array}$$

Using field-read-exc-axiom on the postcondition of *mfg*, we get the following proof outline:

$\{S(y) = \texttt{null}\ \wedge\ \pi(S, c)\ \wedge\ wtp(S, c)\}$
$\Rightarrow$ $[[defsem, swis]]$
$\{S(y) = \texttt{null}\ \wedge\ \pi(S, c)\ \wedge\ wtp(S, c)$
$\wedge\ \exists SQ : ssem(S, c, SQ)\ \wedge\ SQ = S[\$:= S(\$)\langle \texttt{NullPExc} \rangle, \texttt{exc} := new(S(\$), \texttt{NullPExc})]\}$
$\Rightarrow$
$\{y = \texttt{null}\ \wedge\ \pi(S, c)\ \wedge\ \exists SQ : sem(S, c, SQ)\ \wedge\ \bigwedge_{v \in vis(c)\ \{\texttt{exc}\}} SQ(v) = S(v)$
$\wedge\ SQ(\texttt{exc}) = new(S(\$), \texttt{NullPExc})\ \wedge\ SQ(\$) = S(\$)\langle \texttt{NullPExc} \rangle\}$
$\Rightarrow$

$$\left\{ \begin{array}{l} y = \texttt{null}\ \wedge\ \exists SQ : sem(S, c, SQ) \\ \wedge\ \bigwedge_{v \in vis(c)\ \{\texttt{exc}\}} SQ(v) = v \\ \wedge\ SQ(\texttt{exc}) = new(\$, \texttt{NullPExc}) \\ \wedge\ SQ(\$) = \$\langle \texttt{NullPExc} \rangle \end{array} \right\} x = y.a; \{\ \exists SQ : sem(S, c, SQ)\ \wedge\ \pi(SQ, c)\ \}$$

QED fread.

fwrite(VarId FieldId VarId): $c == x.a = e;$

To show:

$$\mathcal{AVI} \models \ \{\ \pi(S,c)\ \}\, x.a = e; \{\ \exists SQ : sem(S,c,SQ)\ \wedge\ \pi(SQ,c)\ \}$$

We show: a)

$$\mathcal{AVI} \models \ \{\ S(x) \neq \texttt{null}\ \wedge\ \pi(S,c)\ \wedge\ wtp(S,c)\ \}\, x.a = e; \{\ \exists SQ : sem(S,c,SQ)\ \wedge\ \pi(SQ,c)\ \}$$

and b)

$$\mathcal{AVI} \models \ \{\ S(x) = \texttt{null}\ \wedge\ \pi(S,c)\ \wedge\ wtp(S,c)\ \}\, x.a = e; \{\ \exists SQ : sem(S,c,SQ)\ \wedge\ \pi(SQ,c)\ \}$$

Then the disjunct-rule and *lm.wtp* allows to derive q.e.d.

AD a:

The property of *ssem*:

$$S(x) \neq \texttt{null} \Rightarrow ssem(S,c,S[\$:= S(\$)\langle S(x).a := e[S]\rangle])$$

Using field-write-axiom on the postcondition of *mfg*, we get the following proof outline:

$\{S(x) \neq \texttt{null}\ \wedge\ \pi(S,c)\ \wedge\ wtp(S,c)\}$
$\Rightarrow$ $[[defofssemforc, swis, lm.wtp]]$
$\{S(x) \neq \texttt{null}\ \wedge\ \pi(S,c)\ \wedge\ wtp(S,c)\ \wedge\ ssem(S,c,S[\$:= S(\$)\langle S(x).a := e[S]\rangle])\}$
$\Rightarrow$
$\{S(x) \neq \texttt{null}\ \wedge\ \pi(S,c)\ \wedge\ \exists SQ : sem(S,c,SQ)\ \wedge\ SQ = S[\$:= S(\$)\langle S(x).a := e[S]\rangle]\}$
$\Rightarrow$
$\{S(x) \neq \texttt{null}\ \wedge\ \pi(S,c)\ \wedge\ \exists SQ : sem(S,c,SQ)\ \wedge\ \bigwedge_{v \in vis(c)} SQ(v) = S(v)$
$\wedge\ SQ(\$) = S(\$)\langle S(x).a := e[S\}\rangle\}$
$\Rightarrow$ $[[lm.ev : \pi(S,c) \Rightarrow e = e[S], strength]]$

$$\left\{ \begin{array}{l} x \neq \texttt{null}\ \wedge\ \exists SQ : sem(S,c,SQ) \\ \wedge\ \bigwedge_{v \in vis(c)} SQ(v) = v \\ \wedge\ SQ(\$) = \$\langle x.a := e\rangle \end{array} \right\} x.a = e; \left\{ \begin{array}{l} \exists SQ : sem(S,c,SQ) \\ \wedge\ \pi(SQ,c) \end{array} \right\}$$

AD b:

The property of *ssem*:

$$S(x) = \texttt{null} \Rightarrow ssem(S,c,S[\$:= S(\$)\langle \texttt{NullPExc}\rangle, \texttt{exc} := new(S(\$), \texttt{NullPExc})])$$

Using field-write-exc-axiom on the postcondition of *mfg*, we get the following proof outline:

$\{S(x) = \texttt{null}\ \wedge\ \pi(S,c)\ \wedge\ wtp(S,c)\}$
$\Rightarrow$ $[[defsem, swis]]$
$\{S(x) = \texttt{null}\ \wedge\ \pi(S,c)\ \wedge\ wtp(S,c)\ \wedge\ \exists SQ : ssem(S,c,SQ)$
$\wedge\ SQ = S[\$:= S(\$)\langle \texttt{NullPExc}\rangle, \texttt{exc} := new(S(\$), \texttt{NullPExc})]\}$
$\Rightarrow$

$$\left\{ \begin{array}{l} x = \texttt{null}\ \wedge\ (\exists SQ : ssem(S,c,SQ) \\ \wedge\ \bigwedge_{v \in vis(c)\ \{\texttt{exc}\}} SQ(v) = v) \\ \wedge\ SQ(\texttt{exc}) = new(\$, \texttt{NullPExc}) \\ \wedge\ SQ(\$) = \$\langle \texttt{NullPExc}\rangle \end{array} \right\} x.a = e; \left\{ \begin{array}{l} \exists SQ : sem(S,c,SQ) \\ \wedge\ \pi(SQ,c) \end{array} \right\}$$

QED fwrite.

cassign(VarId Type Exp): $c == x = (T)e;$

To show:

$$\mathcal{AVI} \models \ \{\ \pi(S,c)\ \}\, x = (T)e; \{\ \exists SQ : sem(S,c,SQ)\ \wedge\ \pi(SQ,c)\ \}$$

We show: a)

$$\mathcal{AVI} \models \ \{\ \tau(e) \preceq T\ \wedge\ \pi(S,c)\ \wedge\ wtp(S,c)\ \}\, c \{\ \exists SQ : sem(S,c,SQ)\ \wedge\ \pi(SQ,c)\ \}$$

and b)

$$\mathcal{AVI} \quad \vdash \quad \left\{ \begin{array}{l} \tau(e) \not\preceq T \\ \ \wedge\ \pi(S,c)\ \wedge\ wtp(S,c) \end{array} \right\} c \left\{ \begin{array}{l} \exists SQ : sem(S,c,SQ) \\ \ \wedge\ \pi(SQ,c) \end{array} \right\}$$

Then disjunct-rule and *lm.wtp* yield the result.

AD a:

The property of *ssem*:

$$\tau(e[S]) \preceq T \Rightarrow ssem(S, x = (T)e;, S[x := e[S]])$$

Using cast-axiom on mfg-postcondition, we get the following proof outline:

$\{\tau(e) \preceq T\ \wedge\ \pi(S,c)\ \wedge\ wtp(S,c)\}$
$\Rightarrow$ $[[ssemp]]$
$\{\tau(e) \preceq T\ \wedge\ \pi(S,c)\ \wedge\ wtp(S,c)\ \wedge\ \tau(e[S]) \preceq T\ \wedge\ \exists SQ : ssem(S,c,SQ)$
$\ \wedge\ SQ = S[x := e[S]]\}$
$\Rightarrow$

$$\left\{ \begin{array}{l} \tau(e) \preceq T\ \wedge\ \exists SQ : sem(S,c,SQ) \\ \ \wedge\ \bigwedge_{v \in vis(c)\ \{x\}} SQ(v) = v \\ \ \wedge\ SQ(x) = e\ \wedge\ SQ(\$) = \$ \end{array} \right\} x = (T)e; \left\{ \begin{array}{l} \exists SQ : sem(S,c,SQ) \\ \ \wedge\ \pi(SQ,c) \end{array} \right\}$$

AD b:

The property of *ssem*:

$$\tau(e[S]) \not\preceq T \Rightarrow ssem(S, c, S[\$:= S(\$)\langle CastExc\rangle, \texttt{exc} := new(S(\$), CastExc)])$$

Using cast-exc-axiom on mfg-postcondition, we get the following proof outline:

$\{\tau(e) \not\preceq T\ \wedge\ \pi(S,c)\ \wedge\ wtp(S,c)\}$
$\Rightarrow$
$\{\tau(e) \not\preceq T\ \wedge\ \pi(S,c)\ \wedge\ wtp(S,c)\ \wedge\ \exists SQ : ssem(S,c,SQ)$
$\ \wedge\ SQ = S[\$:= S(\$)[\$\langle \texttt{CastExc}\rangle/\$, new(\$, \texttt{CastExc})/\texttt{exc}], \texttt{exc} := new(S(\$), CastExc)]\}$
$\Rightarrow$

$$\left\{ \begin{array}{l} \tau(e) \not\preceq T\ \wedge\ \exists SQ : sem(S,c,SQ) \\ \ \wedge\ \bigwedge_{v \in vis(c)\ \{\texttt{exc}\}} SQ(v) = v \\ \ \wedge\ SQ(\texttt{exc}) := new(\$, CastExc) \\ \ \wedge\ SQ(\$) = \$[\$\langle \texttt{CastExc}\rangle/\$, new(\$, \texttt{CastExc})/\texttt{exc}] \end{array} \right\} x = (T)e; \left\{ \begin{array}{l} \exists SQ : sem(S,c,SQ) \\ \ \wedge\ \pi(SQ,c) \end{array} \right\}$$

QED cassign

new(VarId CTypeId): $c == x = \texttt{new}\ T();$

The property of *ssem*:

$$\exists SQ : ssem(S, x = newT();, SQ)\ \wedge\ SQ = S[x := new(S(\$), T), \$:= S(\$)\langle T\rangle]$$

Using new-axiom on the *mgf*-postcondition yields:

$\{\pi(S,c)\ \wedge\ wtp(S,c)\}$
$\Rightarrow$ $[[swis, ssemp]]$
$\{\pi(S,c)\ \wedge\ wtp(S,c)\ \wedge\ \exists SQ : ssem(S,c,SQ)$
$\ \wedge\ SQ = S[x := new(S(\$), T), \$:= S(\$)\langle T\rangle]\}$
$\Rightarrow$

$$\left\{ \begin{array}{l} \exists SQ : sem(S,c,SQ) \\ \ \wedge\ \bigwedge_{v \in vis(c)\ \{x\}} SQ(v) = v \\ \ \wedge\ SQ(x) = new(\$, T) \\ \ \wedge\ SQ(\$) = \$\langle T\rangle \end{array} \right\} x = \texttt{new}\ T(); \left\{ \begin{array}{l} \exists SQ : sem(S,c,SQ) \\ \ \wedge\ \pi(SQ,c) \end{array} \right\}$$

seq(Statement Statement):

In a *seq*-statement it holds $\pi(S,s) = \pi(S,s1) = \pi(S,s2)$. Thus, we may assume

$$\mathcal{AVI} \quad \models \quad \{\ \pi(S,s)\ \}\, s1\, \{\ \exists SQ : sem(S,s1,SQ)\ \wedge\ \pi(SQ,s)\ \}$$
$$\mathcal{AVI} \quad \models \quad \{\ \pi(S,s)\ \}\, s2\, \{\ \exists SQ : sem(S,s2,SQ)\ \wedge\ \pi(SQ,s)\ \}$$

and have to show

$$\mathcal{AVI} \quad \models \quad \{\ \pi(S,s)\ \}\, s\, \{\ \exists SQ : sem(S,s,SQ)\ \wedge\ \pi(SQ,s)\ \}$$

From

$$\mathcal{AVI} \quad \models \quad \{\ \pi(S,s)\ \}\, s1\, \{\ \exists SQ : sem(S,s1,SQ)\ \wedge\ \pi(SQ,s)\ \}$$

we derive the triples (cp. equivalence of MGF with MGF_t and MGF_d):

$$\text{t)}\ \mathcal{AVI} \quad \models \quad \{\ \pi(S,s)\ \wedge\ sem(S,s1,SR)\ \}\, s1\, \{\ \pi(SR,s)\ \}$$
$$\text{d)}\ \mathcal{AVI} \quad \models \quad \{\ \pi(S,s)\ \wedge\ \neg\exists SQ : sem(S,s1,SQ)\ \}\, s1\, \{\ false\ \}$$

Below, we consider two cases. Starting from (t) we derive in case (a)

$$\left\{\begin{array}{l} \pi(S,s)\ \wedge\ sem(S,s1,SR) \\ \quad \wedge\ SR(\texttt{exc}) = \texttt{null} \end{array}\right\} s \left\{\begin{array}{l} \exists SQ : sem(S,s,SQ) \\ \quad \wedge\ \pi(SQ,s) \end{array}\right\}$$

and in case (b)

$$\left\{\begin{array}{l} \pi(S,s)\ \wedge\ sem(S,s1,SR) \\ \quad \wedge\ SR(\texttt{exc}) \neq \texttt{null} \end{array}\right\} s \left\{\begin{array}{l} \exists SQ : sem(S,s,SQ) \\ \quad \wedge\ \pi(SQ,s) \end{array}\right\}$$

Then, disjunction yields the following:

$$\frac{\left\{\begin{array}{l} \pi(S,s) \\ \quad \wedge\ sem(S,s1,SR) \end{array}\right\} s \left\{\begin{array}{l} \exists SQ : sem(S,s,SQ) \\ \quad \wedge\ \pi(SQ,s) \end{array}\right\}}{\left\{\begin{array}{l} \pi(S,s) \\ \quad \wedge\ \exists SR : sem(S,s1,SR) \end{array}\right\} s \left\{\begin{array}{l} \exists SQ : sem(S,s,SQ) \\ \quad \wedge\ \pi(SQ,s) \end{array}\right\}}\ [\text{ex}]$$

Starting with (d), we get

$$\frac{\left\{\begin{array}{l} \pi(S,s) \\ \quad \wedge\ \neg\exists SR : sem(S,s1,SR) \end{array}\right\} s1 \{\ false\ \}}{\left\{\begin{array}{l} \pi(S,s) \\ \quad \wedge\ \neg\exists SR : sem(S,s1,SR) \end{array}\right\} s1 \left\{\begin{array}{l} \exists SQ : sem(S,s,SQ) \\ \quad \wedge\ \pi(SQ,s)\ \wedge\ \texttt{exc} \neq \texttt{null} \end{array}\right\}}\ [\text{weak}]$$

$$\frac{\left\{\begin{array}{l} \pi(S,s) \\ \quad \wedge\ \neg\exists SR : sem(S,s1,SR) \end{array}\right\} s1 \left\{\begin{array}{l} \exists SQ : sem(S,s,SQ) \\ \quad \wedge\ \pi(SQ,s)\ \wedge\ \texttt{exc} \neq \texttt{null} \end{array}\right\}}{\left\{\begin{array}{l} \pi(S,s) \\ \quad \wedge\ \neg\exists SR : sem(S,s1,SR) \end{array}\right\} s \left\{\begin{array}{l} \exists SQ : sem(S,s,SQ) \\ \quad \wedge\ \pi(SQ,s) \end{array}\right\}}\ [\text{seq-exc}]$$

and by disjunction with the above we obtain MGF for s:

$$\{\ \pi(S,s)\ \}\, s\, \{\ \exists SQ : sem(S,s,SQ)\ \wedge\ \pi(SQ,s)\ \}$$

case a:

The semantic property:

$$ssem(S,s1,SR)\ \wedge\ SR(\texttt{exc}) = \texttt{null}\ \wedge\ ssem(SR,s2,SQ) \Rightarrow ssem(S,s1s2,SQ)$$

which implies – using lemma well-typed-res –

$$sem(S,s1,SR)\ \wedge\ SR(\texttt{exc}) = \texttt{null}\ \wedge\ sem(SR,s2,SQ) \Rightarrow sem(S,s1s2,SQ)$$

$$\left\{ \begin{array}{l} \pi(S,s) \\ \quad \wedge\ sem(S,s1,SR) \end{array} \right\} s1 \left\{ \begin{array}{l} \pi(SR,s) \end{array} \right\}$$

[inva]

$$\left\{ \begin{array}{l} \pi(S,s) \\ \quad \wedge\ sem(S,s1,SR) \\ \quad \wedge\ SR(\texttt{exc}) = \texttt{null} \end{array} \right\} s1 \left\{ \begin{array}{l} \pi(SR,s) \\ \quad \wedge\ SR(\texttt{exc}) = \texttt{null} \end{array} \right\}$$

[swis,hypo]

$$\left\{ \begin{array}{l} \pi(S,s) \\ \quad \wedge\ sem(S,s1,SR) \\ \quad \wedge\ SR(\texttt{exc}) = \texttt{null} \end{array} \right\} s1 \left\{ \begin{array}{l} \pi(SR,s) \\ \quad \wedge\ \texttt{exc} = \texttt{null} \end{array} \right\}$$
$$\left\{ \begin{array}{l} \pi(SR,s) \end{array} \right\} s2 \left\{ \begin{array}{l} \exists SQ : sem(SR,s2,SQ) \\ \quad \wedge\ \pi(SQ,s) \end{array} \right\}$$

[seq]

$$\left\{ \begin{array}{l} \pi(S,s) \\ \quad \wedge\ sem(S,s1,SR) \\ \quad \wedge\ SR(\texttt{exc}) = \texttt{null} \end{array} \right\} s \left\{ \begin{array}{l} \exists SQ : sem(SR,s2,SQ) \\ \quad \wedge\ \pi(SQ,s) \end{array} \right\}$$

[inva]

$$\left\{ \begin{array}{l} \pi(S,s) \\ \quad \wedge\ sem(S,s1,SR) \\ \quad \wedge\ SR(\texttt{exc}) = \texttt{null} \end{array} \right\} s \left\{ \begin{array}{l} sem(S,s1,SR) \ \wedge\ SR(\texttt{exc}) = \texttt{null} \\ \quad \wedge\ \exists SQ : sem(SR,s2,SQ) \\ \quad \wedge\ \pi(SQ,s) \end{array} \right\}$$

[weak]

$$\left\{ \begin{array}{l} \pi(S,s) \\ \quad \wedge\ sem(S,s1,SR) \\ \quad \wedge\ SR(\texttt{exc}) = \texttt{null} \end{array} \right\} s \left\{ \begin{array}{l} \exists SQ : sem(S,s1,SR) \ \wedge\ SR(\texttt{exc}) = \texttt{null} \\ \quad \wedge\ sem(SR,s2,SQ) \\ \quad \wedge\ \pi(SQ,s) \end{array} \right\}$$

[ssemp,weak]

$$\left\{ \begin{array}{l} \pi(S,s) \\ \quad \wedge\ sem(S,s1,SR) \\ \quad \wedge\ SR(\texttt{exc}) = \texttt{null} \end{array} \right\} s \left\{ \begin{array}{l} \exists SQ : sem(S,s,SQ) \\ \quad \wedge\ \pi(SQ,s) \end{array} \right\}$$

case b:

The used semantic property derived from def *ssem*:

$$sem(S,s1,SR) \ \wedge\ SR(\texttt{exc}) \neq \texttt{null} \Rightarrow sem(S,s1s2,SR)$$

$$\left\{ \ \pi(S,s) \ \wedge\ sem(S,s1,SR) \ \right\} s1 \left\{ \ \pi(SR,s) \ \right\}$$

[inva]

$$\left\{ \begin{array}{l} \pi(S,s) \ \wedge\ sem(S,s1,SR) \\ \quad \wedge\ SR(\texttt{exc}) \neq \texttt{null} \end{array} \right\} s1 \left\{ \begin{array}{l} \pi(SR,s) \\ \quad \wedge\ SR(\texttt{exc}) \neq \texttt{null} \end{array} \right\}$$

[seq-exc]

$$\left\{ \begin{array}{l} \pi(S,s) \ \wedge\ sem(S,s1,SR) \\ \quad \wedge\ SR(\texttt{exc}) \neq \texttt{null} \end{array} \right\} s \left\{ \ \pi(SR,s) \ \right\}$$

[inva]

$$\left\{ \begin{array}{l} \pi(S,s) \ \wedge\ sem(S,s1,SR) \\ \quad \wedge\ SR(\texttt{exc}) \neq \texttt{null} \end{array} \right\} s \left\{ \begin{array}{l} \pi(SR,s) \ \wedge\ sem(S,s1,SR) \\ \quad \wedge\ SR(\texttt{exc}) \neq \texttt{null} \end{array} \right\}$$

[semp,weak]

$$\left\{ \begin{array}{l} \pi(S,s) \ \wedge\ sem(S,s1,SR) \\ \quad \wedge\ SR(\texttt{exc}) \neq \texttt{null} \end{array} \right\} s \left\{ \begin{array}{l} \pi(SR,s) \\ \quad \wedge\ sem(S,s,SR) \end{array} \right\}$$

[weak]

$$\left\{\begin{array}{l} \pi(S,s) \ \wedge \ sem(S,s1,SR) \\ \ \wedge \ SR(\texttt{exc}) \neq \texttt{null} \end{array}\right\} s \left\{\begin{array}{l} \exists SQ : sem(S,s,SQ) \\ \ \wedge \ \pi(SQ,s) \end{array}\right\}$$

QED *seq*

`if` (Exp Statement Statement): $s == \texttt{if}(e)\{s1\}\texttt{else}\{s2\}$

The induction hypo yields

$$\mathcal{AVI} \ \models \ \{ \ \pi(S,s) \ \} \, s1 \, \{ \ \exists SQ : sem(S,s1,SQ) \ \wedge \ \pi(SQ,s) \ \}$$
$$\mathcal{AVI} \ \models \ \{ \ \pi(S,s) \ \} \, s2 \, \{ \ \exists SQ : sem(S,s2,SQ) \ \wedge \ \pi(SQ,s) \ \}$$

from which we show

$$\mathcal{AVI} \ \models \ \{ \ \pi(S,s) \ \} \, s \, \{ \ \exists SQ : sem(S,s,SQ) \ \wedge \ \pi(SQ,s) \ \}$$

$$\frac{\frac{\frac{\frac{\frac{\{ \ \pi(S,s) \ \} \, s1 \, \{ \ \exists SQ : sem(S,s1,SQ) \ \wedge \ \pi(SQ,s) \ \}}{\{ \ e[S] \ \wedge \ \pi(S,s) \ \} \, s1 \, \{ \ \exists SQ : e[S] \ \wedge \ sem(S,s1,SQ) \ \wedge \ \pi(SQ,s) \ \}}\,[\text{inva,weak}]}{\{ \ e \ \wedge \ \pi(S,s) \ \} \, s1 \, \{ \ \exists SQ : e[S] \ \wedge \ sem(S,s1,SQ) \ \wedge \ \pi(SQ,s) \ \}}\,[\text{inva,strg}]}{\{ \ e \ \wedge \ \pi(S,s) \ \} \, s1 \, \{ \ \exists SQ : e[S] \ \wedge \ wtp(S,s) \ \wedge \ ssem(S,s1,SQ) \ \wedge \ \pi(SQ,s) \ \}}\,[\text{def } sem]}{\{ \ e \ \wedge \ \pi(S,s) \ \} \, s1 \, \{ \ \exists SQ : wtp(S,s) \ \wedge \ ssem(S,s,SQ) \ \wedge \ \pi(SQ,s) \ \}}\,[\text{ssemp,weak}]}{\{ \ e \ \wedge \ \pi(S,s) \ \} \, s1 \, \{ \ \exists SQ : sem(S,s,SQ) \ \wedge \ \pi(SQ,s) \ \}}\,[\text{def } sem]$$

In the same way, we derive:

$$\frac{\frac{\frac{\frac{\frac{\{ \ \pi(S,s) \ \} \, s2 \, \{ \ \exists SQ : sem(S,s2,SQ) \ \wedge \ \pi(SQ,s) \ \}}{\{ \ \neg e[S] \ \wedge \ \pi(S,s) \ \} \, s2 \, \{ \ \exists SQ : \neg e[S] \ \wedge \ sem(S,s2,SQ) \ \wedge \ \pi(SQ,s) \ \}}\,[\text{inva,weak}]}{\{ \ \neg e \ \wedge \ \pi(S,s) \ \} \, s2 \, \{ \ \exists SQ : \neg e[S] \ \wedge \ sem(S,s2,SQ) \ \wedge \ \pi(SQ,s) \ \}}\,[\text{inva,strg}]}{\{ \ \neg e \ \wedge \ \pi(S,s) \ \} \, s2 \, \{ \ \exists SQ : \neg e[S] \ \wedge \ wt(S,s) \ \wedge \ ssem(S,s2,SQ) \ \wedge \ \pi(SQ,s) \ \}}\,[\text{def } sem]}{\{ \ \neg e \ \wedge \ \pi(S,s) \ \} \, s2 \, \{ \ \exists SQ : wt(S,s) \ \wedge \ ssem(S,s,SQ) \ \wedge \ \pi(SQ,s) \ \}}\,[\text{ssemp,weak}]}{\{ \ \neg e \ \wedge \ \pi(S,s) \ \} \, s2 \, \{ \ \exists SQ : sem(S,s,SQ) \ \wedge \ \pi(SQ,s) \ \}}\,[\text{def } sem]$$

and the application of the `if`-rule yields the desired result.
QED `if`

`while` (Exp Statement): $c == \texttt{while}(e)\{s\}$

We show mgf_t and mgf_d:

AD t:

assuming

$$\mathcal{AVI} \ \models \ \{ \ sem(S,s,SQ) \ \wedge \ \pi(S,s) \ \} \, s \, \{ \ \pi(SQ,s) \ \}$$

show

$$\mathcal{AVI} \ \models \ \{ \ sem(S,w,SQ) \ \wedge \ \pi(S,w) \ \} \, w \, \{ \ \pi(SQ,w) \ \}$$

The semantics of the `while` loop is:

$$\begin{aligned} sem(SP, w(e)\{s\}, SQ) \quad \Leftrightarrow \quad & wtp(SP, w) \ \wedge \ ((\neg e[SP] \ \wedge \ SP = SQ) \\ & \vee \ (e[SP] \ \wedge \ ssem(SP, s, SQ) \ \wedge \ SQ(\texttt{exc}) \neq \texttt{null}) \\ & \vee \ (e[SP] \ \wedge \ (\exists SR : ssem(SP, s, SR) \\ & \qquad \wedge \ SR(\texttt{exc}) = \texttt{null} \ \wedge \ ssem(SR, w, SQ)))) \end{aligned}$$

Let

$$inv(SP, w(e)\{s\}, SQ) \quad \Leftrightarrow \quad sem(SP, w(e)\{s\}, SQ) \vee (SP = SQ \ \wedge \ SP(\texttt{exc}) \neq \texttt{null})$$

We use the following loop invariant:

$$\exists S \quad : \quad \pi(S, w) \ \wedge \ inv(S, w(e)\{s\}, SQ)$$

i.e. the current state S is a state leading to a terminating loop state SQ or it is a state with an exception (the invariant has to hold after an abnormal termination of the loop).

The proof starts with the induction hypothesis about the loop body (for brevity, we omit the assumptions):

$$\frac{\left\{ \ sem(SP, s, SQ) \ \wedge \ \pi(SP, w) \ \right\} s \left\{ \ \pi(SQ, w) \ \right\}}{\left\{ \begin{array}{l} sem(SP, s, SQ) \ \wedge \ inv(SQ, w, SR) \\ \quad \wedge \ \pi(SP, w) \end{array} \right\} s \left\{ \begin{array}{l} inv(SQ, w, SR) \\ \quad \wedge \ \pi(SQ, w) \end{array} \right\}} \ [\text{inva}]$$

$$\frac{\left\{ \begin{array}{l} sem(SP, s, SQ) \ \wedge \ inv(SQ, w, SR) \\ \quad \wedge \ \pi(SP, w) \end{array} \right\} s \left\{ \begin{array}{l} inv(SQ, w, SR) \\ \quad \wedge \ \pi(SQ, w) \end{array} \right\}}{\left\{ \begin{array}{l} \exists SQ : sem(SP, s, SQ) \ \wedge \ inv(SQ, w, SR) \\ \quad \wedge \ \pi(SP, w) \end{array} \right\} s \left\{ \begin{array}{l} \exists SP : inv(SP, w, SR) \\ \quad \wedge \ \pi(SP, w) \end{array} \right\}} \ [\text{weak,ex}]$$

$$\frac{\left\{ \begin{array}{l} \exists SQ : sem(SP, s, SQ) \ \wedge \ inv(SQ, w, SR) \\ \quad \wedge \ \pi(SP, w) \end{array} \right\} s \left\{ \begin{array}{l} \exists SP : inv(SP, w, SR) \\ \quad \wedge \ \pi(SP, w) \end{array} \right\}}{\left\{ \begin{array}{l} e[SP] \ \wedge \ SP(\texttt{exc}) = \texttt{null} \ \wedge \ inv(SP, w, SR) \\ \quad \wedge \ \pi(SP, w) \end{array} \right\} s \left\{ \begin{array}{l} \exists SP : inv(SP, w, SR) \\ \quad \wedge \ \pi(SP, w) \end{array} \right\}} \ [\text{strg,lm.w1}]$$

$$\frac{\left\{ \begin{array}{l} e[SP] \ \wedge \ SP(\texttt{exc}) = \texttt{null} \ \wedge \ inv(SP, w, SR) \\ \quad \wedge \ \pi(SP, w) \end{array} \right\} s \left\{ \begin{array}{l} \exists SP : inv(SP, w, SR) \\ \quad \wedge \ \pi(SP, w) \end{array} \right\}}{\left\{ \begin{array}{l} e \ \wedge \ \exists SP : inv(SP, w, SR) \\ \quad \wedge \ \pi(SP, w) \end{array} \right\} s \left\{ \begin{array}{l} \exists SP : inv(SP, w, SR) \\ \quad \wedge \ \pi(SP, w) \end{array} \right\}} \ [\text{swis,ex}]$$

$$\frac{\left\{ \begin{array}{l} e \ \wedge \ \exists SP : inv(SP, w, SR) \\ \quad \wedge \ \pi(SP, w) \end{array} \right\} s \left\{ \begin{array}{l} \exists SP : inv(SP, w, SR) \\ \quad \wedge \ \pi(SP, w) \end{array} \right\}}{\left\{ \begin{array}{l} \exists SP : inv(SP, w, SR) \\ \quad \wedge \ \pi(SP, w) \end{array} \right\} \texttt{while}(e)\{s\} \left\{ \begin{array}{l} (\texttt{exc} \neq \texttt{null} \vee \neg e) \\ \quad \wedge \ (\exists SP : inv(SP, w, SR) \\ \qquad \wedge \ \pi(SP, w)) \end{array} \right\}} \ [\texttt{while}]$$

$$\frac{\left\{ \begin{array}{l} \exists SP : inv(SP, w, SR) \\ \quad \wedge \ \pi(SP, w) \end{array} \right\} \texttt{while}(e)\{s\} \left\{ \begin{array}{l} \exists SP : (SP(\texttt{exc}) \neq \texttt{null} \\ \qquad \vee \ \neg e[SP]) \\ \quad \wedge \ inv(SP, w, SR) \\ \quad \wedge \ \pi(SP, w) \end{array} \right\}}{\left\{ \begin{array}{l} \exists SP : inv(SP, w, SR) \\ \quad \wedge \ \pi(SP, w) \end{array} \right\} \texttt{while}(e)\{s\} \left\{ \begin{array}{l} \exists SP : SP = SR \\ \quad \wedge \ \pi(SR, w) \end{array} \right\}} \ [\text{weak}]$$

$$\frac{\left\{ \begin{array}{l} \exists SP : inv(SP, w, SR) \\ \quad \wedge \ \pi(SP, w) \end{array} \right\} \texttt{while}(e)\{s\} \left\{ \begin{array}{l} \exists SP : SP = SR \\ \quad \wedge \ \pi(SR, w) \end{array} \right\}}{\left\{ \ inv(SP, w, SR) \ \wedge \ \pi(SP, w) \ \right\} \texttt{while}(e)\{s\} \left\{ \ \pi(SR, w) \ \right\}} \ [\text{lm.w2}]$$

$$\frac{\left\{ \ inv(SP, w, SR) \ \wedge \ \pi(SP, w) \ \right\} \texttt{while}(e)\{s\} \left\{ \ \pi(SR, w) \ \right\}}{\left\{ \ sem(SP, w, SR) \ \wedge \ \pi(SP, w) \ \right\} \texttt{while}(e)\{s\} \left\{ \ \pi(SR, w) \ \right\}} \ [\text{swis}]$$

$$[\text{strg}]$$

Lemma 17 (w1)**.**

$$e[S] \ \wedge \ S(\texttt{exc}) = \texttt{null} \ \wedge \ inv(S, w, SQ) \quad \Rightarrow \quad \exists SR : sem(S, s, SR) \ \wedge \ inv(SR, w, SQ)$$

PROOF:

$e[S] \ \wedge \ S(\texttt{exc}) = \texttt{null} \ \wedge \ inv(S, w, SQ)$

$\Rightarrow$

$e[S] \ \wedge \ S(\texttt{exc}) = \texttt{null} \ \wedge \ (sem(S, w(e)\{s\}, SQ) \vee (S = SQ \ \wedge \ S(\texttt{exc}) \neq \texttt{null}))$

$\Rightarrow$

$e[S] \ \wedge \ S(\texttt{exc}) = \texttt{null} \ \wedge \ wtp(S, w)$
$\wedge \ ((\neg e[S] \ \wedge \ S = SQ)$
$\vee \ (e[S] \ \wedge \ ssem(S, s, SQ) \ \wedge \ SQ(\texttt{exc}) \neq \texttt{null})$
$\vee \ (e[S] \ \wedge \ (\exists SR : ssem(S, s, SR) \ \wedge \ SR(\texttt{exc}) = \texttt{null} \ \wedge \ ssem(SR, w, SQ))))$

$\Rightarrow$

$wtp(S, w) \ \wedge \ ((ssem(S, s, SQ) \ \wedge \ SQ(\texttt{exc}) \neq \texttt{null})$
$\vee \ (\exists SR : ssem(S, s, SR) \ \wedge \ SR(\texttt{exc}) = \texttt{null} \ \wedge \ ssem(SR, w, SQ)))$

$\Rightarrow \ [[well - typed - res]]$

$(sem(S, s, SQ) \ \wedge \ SQ(\texttt{exc}) \neq \texttt{null})$
$\vee \ (\exists SR : sem(S, s, SR) \ \wedge \ wtp(SR, w) \ \wedge \ ssem(SR, w, SQ))$

$\Rightarrow$

$(\exists SR : sem(S, s, SR) \ \wedge \ SR = SQ \ \wedge \ SR(\texttt{exc}) \neq \texttt{null})$
$\vee \ (\exists SR : sem(S, s, SR) \ \wedge \ sem(SR, w, SQ))$

$\Rightarrow \ [[def \ inv]]$

$EX \ SR : sem(S, s, SR) \ \wedge \ inv(SR, w, SQ)$

QED $lm.w1$

Lemma 18 (w2)**.**

$$(SP(\texttt{exc}) \neq \texttt{null} \vee \neg e[SP]) \ \wedge \ inv(SP, w, SQ) \quad \Rightarrow \quad SP = SQ$$

PROOF:

$(SP(\texttt{exc}) \neq \texttt{null} \vee \neg e[SP])$
$\wedge \ (sem(SP, w(e)\{s\}, SQ) \vee (SP = SQ \ \wedge \ SP(\texttt{exc}) \neq \texttt{null}))$

$\Rightarrow$

$(SP(\texttt{exc}) \neq \texttt{null} \ \wedge \ sem(SP, w(e)\{s\}, SQ))$
$\vee \ (SP(\texttt{exc}) \neq \texttt{null} \ \wedge \ SP = SQ \ \wedge \ SP(\texttt{exc}) \neq \texttt{null})$
$\vee \ (\neg e[SP] \ \wedge \ sem(SP, w(e)\{s\}, SQ)) \vee (\neg e[SP] \ \wedge \ SP = SQ \ \wedge \ SP(\texttt{exc}) \neq \texttt{null})$

$\Rightarrow \ [[defsemandssem]]$

$(SP(\texttt{exc}) \neq \texttt{null} \ \wedge \ wtp(SP, w)) \vee (SP(\texttt{exc}) \neq \texttt{null} \ \wedge \ SP = SQ)$
$\vee \ (\neg e[SP] \ \wedge \ (\neg e[SP] \ \wedge \ SP = SQ)) \vee (\neg e[SP] \ \wedge \ SP = SQ)$

$\Rightarrow$

$false \vee SP = SQ \vee SP = SQ \vee SP = SQ$

$\Rightarrow$

$SP = SQ$

QED Lemma $w2$

QED (t)

AD (d):

assuming

$$\mathcal{AVI} \ \models \ \{ \ sem(S, s, SQ) \ \wedge \ \pi(S, s) \ \} \, s \, \{ \ \pi(SQ, s) \ \}$$
$$\mathcal{AVI} \ \models \ \{ \ (\neg \exists SQ : sem(S, s, SQ)) \ \wedge \ \pi(S, s) \ \} \, s \, \{ \ false \ \}$$

show

$$\mathcal{AVI} \ \models \ \{ \ (\neg \exists SQ : sem(S, w, SQ)) \ \wedge \ \pi(S, w) \ \} \, w \, \{ \ false \ \}$$

We use the following invariant:

$$\exists SP : \pi(SP, w) \ \wedge \ wtp(SP, w) \ \wedge \ \neg \exists SQ : sem(SP, w, SQ)$$

which is equivalent to

$$\exists SP : \pi(SP, w) \ \wedge \ wtp(SP, w) \ \wedge \ \neg\exists SQ : inv(SP, w, SQ)$$

The proof starts with the two induction hypotheses about s (for brevity, we omit the assumptions):

From the first hypothesis:

$$\frac{\left\{ \ sem(S,s,SQ) \ \wedge \ \pi(S,s) \ \right\} s \left\{ \ \pi(SQ,s) \ \right\}}{\left\{ \begin{array}{l} sem(S,s,SQ) \ \wedge \ \pi(S,s) \\ \ \wedge \ wtp(SQ,w) \\ \ \wedge \ \neg\exists SR : sem(SQ,w,SR) \end{array} \right\} s \left\{ \begin{array}{l} \pi(SQ,s) \\ \ \wedge \ wtp(SQ,w) \\ \ \wedge \ \neg\exists SR : sem(SQ,w,SR) \end{array} \right\}} \ [\text{inva}]$$

$$\frac{\left\{ \begin{array}{l} sem(S,s,SQ) \ \wedge \ \pi(S,s) \\ \ \wedge \ wtp(SQ,w) \\ \ \wedge \ \neg\exists SR : sem(SQ,w,SR) \end{array} \right\} s \left\{ \begin{array}{l} \exists SP : \pi(SP,s) \\ \ \wedge \ wtp(SP,w) \\ \ \wedge \ \neg\exists SR : sem(SP,w,SR) \end{array} \right\}}{\left\{ \begin{array}{l} sem(S,s,SQ) \ \wedge \ e[S] \ \wedge \ \pi(S,w) \\ \ \wedge \ \neg\exists SR : sem(SQ,w,SR) \\ \ \wedge \ wtp(SQ,w) \end{array} \right\} s \left\{ \begin{array}{l} \exists SP : \pi(SP,s) \\ \ \wedge \ wtp(SP,w) \\ \ \wedge \ \neg\exists SR : sem(SP,w,SR) \end{array} \right\}} \ [\text{strg}]$$

$$\frac{\left\{ \begin{array}{l} sem(S,s,SQ) \ \wedge \ e[S] \ \wedge \ \pi(S,w) \\ \ \wedge \ \neg\exists SR : sem(S,w,SR) \end{array} \right\} s \left\{ \begin{array}{l} \exists SP : \pi(SP,s) \ \wedge \ wtp(SP,w) \\ \ \wedge \ \neg\exists SR : sem(SP,w,SR) \end{array} \right\}}{\left\{ \begin{array}{l} (\exists SQ : sem(S,s,SQ)) \\ \ \wedge \ e \ \wedge \ \pi(S,w) \\ \ \wedge \ \neg\exists SR : sem(S,w,SR) \end{array} \right\} s \left\{ \begin{array}{l} \exists SP : \pi(SP,w) \\ \ \wedge \ wtp(SP,w) \\ \ \wedge \ \neg\exists SR : sem(SP,w,SR) \end{array} \right\}} \ [\text{lm.w3,strg}]$$

$$[\text{ex,strg}]$$

From the second hypothesis:

$$\frac{\left\{ \ \neg(\exists SQ : sem(S,s,SQ)) \ \wedge \ \pi(S,s) \ \right\} s \left\{ \ false \ \right\}}{\left\{ \begin{array}{l} \neg(\exists SQ : sem(S,s,SQ)) \\ \ \wedge \ e \ \wedge \ \pi(S,w) \\ \ \wedge \ \neg\exists SR : sem(S,w,SR) \end{array} \right\} s \left\{ \begin{array}{l} \exists SP : \pi(SP,w) \\ \ \wedge \ wtp(SP,w) \\ \ \wedge \ \neg\exists SR : sem(SP,w,SR) \end{array} \right\}} \ [\text{swis}]$$

By disjunct-rule, the conclusion of the above derivations yield:

$$\frac{\left\{ \begin{array}{l} e \ \wedge \ \pi(S,w) \\ \ \wedge \ \neg\exists SR : sem(S,w,SR) \end{array} \right\} s \left\{ \begin{array}{l} \exists SP : \pi(SP,w) \ \wedge \ wtp(SP,w) \\ \ \wedge \ \neg\exists SR : sem(SP,w,SR) \end{array} \right\}}{\left\{ \begin{array}{l} e \ \wedge \ \exists SP : \pi(SP,w) \\ \ \wedge \ wtp(SP,w) \\ \ \wedge \ \neg\exists SR : sem(SP,w,SR) \end{array} \right\} s \left\{ \begin{array}{l} \exists SP : \pi(SP,w) \\ \ \wedge \ wtp(SP,w) \\ \ \wedge \ \neg\exists SR : sem(SP,w,SR) \end{array} \right\}} \ [\text{strg,ex}]$$

$$[\texttt{while}]$$

$$\left\{\begin{array}{l}\exists SP:\pi(SP,w)\\ \ \wedge\ wtp(SP,w)\\ \ \wedge\ \neg\exists SR:\\ \qquad sem(SP,w,SR)\end{array}\right\}\texttt{while}(e)\{s\}\left\{\begin{array}{l}(\texttt{exc}\neq\texttt{null}\vee\neg e)\\ \ \wedge\ \exists SP:\pi(SP,w)\\ \ \wedge\ wtp(SP,w)\\ \ \wedge\ \neg\exists SR:\\ \qquad sem(SP,w,SR)\end{array}\right\}$$

[weak]

$$\left\{\begin{array}{l}\exists SP:\\ \quad \pi(SP,w)\\ \ \wedge\ wtp(SP,w)\\ \ \wedge\ \neg\exists SR:\\ \qquad sem(SP,w,SR)\end{array}\right\}\texttt{while}(e)\{s\}\left\{\begin{array}{l}\exists SP:\\ \quad (SP(\texttt{exc})\neq\texttt{null}\vee\neg e[SP])\\ \ \wedge\ \pi(SP,w)\ \wedge\ wtp(SP,w)\\ \ \wedge\ \neg\exists SR:\\ \qquad sem(SP,w,SR)\end{array}\right\}$$

[weak,def wtp]

$$\left\{\begin{array}{l}\exists SP:\\ \pi(SP,w)\ \wedge\ wtp(SP,w))\\ \ \wedge\ \neg\exists SR:\\ \qquad sem(SP,w,SR)\end{array}\right\}\texttt{while}(e)\{s\}\left\{\begin{array}{l}\exists SP:\pi(SP,w)\\ \ \wedge\ \neg e[SP]\ \wedge\ wtp(SP,w))\\ \ \wedge\ \neg\exists SR:\\ \qquad sem(SP,w,SR)\end{array}\right\}$$

[semp]

$$\left\{\begin{array}{l}\exists SP:\pi(SP,w)\\ \ \wedge\ wtp(SP,w))\\ \ \wedge\ \neg\exists SR:\\ \qquad sem(SP,w,SR)\end{array}\right\}\texttt{while}(e)\{s\}\left\{\begin{array}{l}\exists SP:\pi(SP,w)\\ \ \wedge\ sem(SP,w,SP))\\ \ \wedge\ \neg\exists SR:\\ \qquad sem(SP,w,SR)\end{array}\right\}$$

[weak]

$$\left\{\begin{array}{l}\exists S:\pi(S,w)\ \wedge\ wtp(S,w))\\ \ \wedge\ \neg\exists SR:sem(S,w,SR)\end{array}\right\}\texttt{while}(e)\{s\}\left\{\ false\ \right\}$$

[strg]

$$\left\{\ \neg\exists SR:sem(S,w,SR)\ \wedge\ \pi(S,w)\ \wedge\ wtp(S,w)\ \right\}\texttt{while}(e)\{s\}\left\{\ false\ \right\}$$

[lm.wtp]

$$\left\{\ \neg\exists SR:sem(S,w,SR)\ \wedge\ \pi(S,w)\ \right\}\texttt{while}(e)\{s\}\left\{\ false\ \right\}$$

Lemma 19 (w3)**.**

$$\begin{array}{l}\quad sem(S,s,SQ)\ \wedge\ e[S]\ \wedge\ \neg(\exists SR:sem(S,w,SR))\\ \Rightarrow\\ \quad \neg\exists SR:sem(SQ,w,SR)\ \wedge\ wtp(SQ,w)\end{array}$$

PROOF:
The lemma is equivalent to:

$$\begin{array}{l}\quad \neg sem(S,s,SQ)\vee\neg e[S]\vee(\exists SR:sem(S,w,SR))\\ \quad \vee(wtp(SQ,w)\ \wedge\ \neg(\exists SR:sem(SQ,w,SR)))\\ \Leftrightarrow\\ \quad \neg sem(S,s,SQ)\vee\neg e[S]\vee(\exists SR:sem(S,w,SR))\\ \quad \vee\neg(\neg wtp(SQ,w)\vee(\exists SR:sem(SQ,w,SR)))\\ \Leftrightarrow\\ \quad (sem(S,s,SQ)\ \wedge\ e[S]\ \wedge\ (\neg wtp(SQ,w)\vee(\exists SR:sem(SQ,w,SR)))\\ \quad \Rightarrow\exists SR:sem(S,w,SR)\end{array}$$

So let us start with the premise:

$sem(S,s,SQ) \ \wedge \ e[S] \ \wedge \ (\neg wtp(SQ,w) \vee (\exists SR : sem(SQ,w,SR)))$

$\Rightarrow$

$(wtp(S,w) \ \wedge \ e[S] \ \wedge \ ssem(S,s,SQ) \ \wedge \ \neg wtp(SQ,w))$
$\vee \ (\exists SR : wtp(S,w) \ \wedge \ e[S] \ \wedge \ ssem(S,s,SQ) \ \wedge \ wtp(SQ,w) \ \wedge \ ssem(SQ,w,SR))$

$\Rightarrow$ $[[lm.well - typed - res]]$

$(wtp(S,w) \ \wedge \ e[S] \ \wedge \ ssem(S,s,SQ) \ \wedge \ wtr(SQ,w) \ \wedge \ \neg wtp(SQ,w))$
$\vee \ (\exists SR : wtp(S,w) \ \wedge \ e[S] \ \wedge \ ssem(S,s,SQ) \ \wedge \ wtp(SQ,w) \ \wedge \ ssem(SQ,w,SR))$

$\Rightarrow$ $[[semp, SQ(\texttt{exc}) \neq \texttt{null} in the first disjunct]]$

$sem(S,w(e)\{s\},SQ) \vee (\exists SR : wtp(S,w) \ \wedge \ e[S] \ \wedge \ ssem(S,s,SQ)$
$\wedge \ SQ(\texttt{exc}) = \texttt{null} \ \wedge \ ssem(SQ,w,SR))$ QED

$\Rightarrow$ $[[PL]]$

$(\exists SR : sem(S,w(e)\{s\},SR)) \vee (\exists SR : wtp(S,w)$
$\wedge \ e[S] \ \wedge \ \exists SQR : ssem(S,s,SQR)$
$\wedge \ SQR(\texttt{exc}) = \texttt{null} \ \wedge \ ssem(SQR,w,SR))$

$\Rightarrow$ $[[semp]]$

$(\exists SR : sem(S,w(e)\{s\},SR)) \vee (\exists SR : sem(S,w(e)\{s\},SR))$

$\Rightarrow$ $[[semp]]$

$\exists SR : sem(S,w,SR)$

Lemma.w3
QED (d)
QED `while`

catch(Statement CTypeId Statement): $s == try\{s0\}catch(Te)\{s1\}$

The induction hypo yields

$$\mathcal{AVI} \ \ \triangleright \ \ \{ \ \pi(S,s) \ \} \, s0 \, \{ \ \exists SQ : sem(S,s0,SQ) \ \wedge \ \pi(SQ,s) \ \}$$
$$\mathcal{AVI} \ \ \triangleright \ \ \{ \ \pi(S,s) \ \} \, s1 \, \{ \ \exists SQ : sem(S,s1,SQ) \ \wedge \ \pi(SQ,s) \ \}$$

from which we have to show

$$\mathcal{AVI} \ \ \triangleright \ \ \{ \ \pi(S,s) \ \} \, s \, \{ \ \exists SQ : sem(S,s,SQ) \ \wedge \ \pi(SQ,s) \ \}$$

From the induction hypo we derive the triples (cf. equivalence of MGF with MGF_t and MGF_d):

$$\text{t)} \ \mathcal{AVI} \ \ \triangleright \ \ \{ \ \pi(S,s) \ \wedge \ sem(S,si,SR) \ \} \, si \, \{ \ \pi(SR,s) \ \}$$
$$\text{d)} \ \mathcal{AVI} \ \ \triangleright \ \ \{ \ \pi(S,s) \ \wedge \ \neg\exists SQ : sem(S,si,SQ) \ \} \, si \, \{ \ false \ \}$$

where si stands for $s0$ or $s1$. On the one hand, starting with (t), we will consider three cases (a)-(c), in which we will derive

$$\left\{ \begin{array}{l} \pi(S,s) \ \wedge \ sem(S,s0,SR) \\ \wedge \ SR(\texttt{exc}) = \texttt{null} \end{array} \right\} s \, \{ \ \exists SQ : sem(S,s,SQ) \ \wedge \ \pi(SQ,s) \ \}$$

$$\left\{ \begin{array}{l} \pi(S,s) \ \wedge \ sem(S,s0,SR) \\ \wedge \ SR(\texttt{exc}) \neq \texttt{null} \\ \wedge \ \tau(SR(\texttt{exc})) \not\preceq T \end{array} \right\} s \left\{ \begin{array}{l} \exists SQ : sem(S,s,SQ) \\ \wedge \ \pi(SQ,s) \end{array} \right\}$$

$$\left\{ \begin{array}{l} \pi(S,s) \ \wedge \ sem(S,s0,SR) \\ \wedge \ SR(\texttt{exc}) \neq \texttt{null} \ \wedge \ \tau(SR(\texttt{exc})) \preceq T \end{array} \right\} s \left\{ \begin{array}{l} \exists SQ : sem(S,s,SQ) \\ \wedge \ \pi(SQ,s) \end{array} \right\}$$

From these results disjunction yields:

$$\frac{\{ \ \pi(S,s) \ \wedge \ sem(S,s0,SR) \ \} \, s \, \{ \ \exists SQ : sem(S,s,SQ) \ \wedge \ \pi(SQ,s) \ \}}{\{ \ \pi(S,s) \ \wedge \ \exists SR : sem(S,s0,SR) \ \} \, s \, \{ \ \exists SQ : sem(S,s,SQ) \ \wedge \ \pi(SQ,s) \ \}} \ [\text{ex}]$$

On the other hand, starting with (d), we get

$$\{ \ \pi(S,s) \ \wedge \ \neg\exists SR : sem(S,s0,SR) \ \} \, s0 \, \{ \ false \ \}$$

$$\frac{}{\left\{\begin{array}{l}\pi(S,s)\\ \ \wedge\ \neg\exists SR : sem(S,s0,SR)\end{array}\right\} s0 \left\{\begin{array}{l}\exists SQ : sem(S,s,SQ)\\ \ \wedge\ \pi(SQ,s)\ \wedge\ \texttt{exc} = \texttt{null}\end{array}\right\}}\ [\text{weak}]$$

$$\frac{\left\{\begin{array}{l}\pi(S,s)\\ \ \wedge\ \neg\exists SR : sem(S,s0,SR)\end{array}\right\} s0 \left\{\begin{array}{l}\exists SQ : sem(S,s,SQ)\\ \ \wedge\ \pi(SQ,s)\ \wedge\ \texttt{exc} = \texttt{null}\end{array}\right\}}{\left\{\begin{array}{l}\pi(S,s)\\ \ \wedge\ \neg\exists SR : sem(S,s0,SR)\end{array}\right\} s \left\{\begin{array}{l}\exists SQ : sem(S,s,SQ)\\ \ \wedge\ \pi(SQ,s)\end{array}\right\}}\ [\text{catch}]$$

and by disjunction MGF for s:

$$\left\{\ \pi(S,s)\ \right\} s \left\{\ \exists SQ : sem(S,s,SQ)\ \wedge\ \pi(SQ,s)\ \right\}$$

case a:

Semantic property:

$$ssem(S,s0,SQ)\ \wedge\ SQ(\texttt{exc}) = \texttt{null}\ \ \Rightarrow\ \ ssem(S,s,SQ)$$

yields

$$sem(S,s0,SQ)\ \wedge\ SQ(\texttt{exc}) = \texttt{null}\ \ \Rightarrow\ \ sem(S,s,SQ)$$

$$\frac{\left\{\ \pi(S,s)\ \wedge\ sem(S,s0,SR)\ \right\} s0 \left\{\ \pi(SR,s)\ \right\}}{\left\{\begin{array}{l}\pi(S,s)\ \wedge\ sem(S,s0,SR)\\ \ \wedge\ SR(\texttt{exc}) = \texttt{null}\end{array}\right\} s0 \left\{\begin{array}{l}sem(S,s0,SR)\ \wedge\ SR(\texttt{exc}) = \texttt{null}\\ \ \wedge\ \pi(SR,s)\end{array}\right\}}\ [\text{inva}]$$

$$\frac{\left\{\begin{array}{l}\pi(S,s)\ \wedge\ sem(S,s0,SR)\\ \ \wedge\ SR(\texttt{exc}) = \texttt{null}\end{array}\right\} s0 \left\{\begin{array}{l}sem(S,s0,SR)\\ \ \wedge\ \pi(SR,s)\ \wedge\ \texttt{exc} = \texttt{null}\end{array}\right\}}{\left\{\begin{array}{l}\pi(S,s)\ \wedge\ sem(S,s0,SR)\\ \ \wedge\ SR(\texttt{exc}) = \texttt{null}\end{array}\right\} s \left\{\begin{array}{l}sem(S,s,SR)\\ \ \wedge\ \pi(SR,s)\end{array}\right\}}\ [\text{semp,weak}]$$

$$\frac{\left\{\begin{array}{l}\pi(S,s)\ \wedge\ sem(S,s0,SR)\\ \ \wedge\ SR(\texttt{exc}) = \texttt{null}\end{array}\right\} s \left\{\begin{array}{l}sem(S,s,SR)\\ \ \wedge\ \pi(SR,s)\end{array}\right\}}{\left\{\begin{array}{l}\pi(S,s)\ \wedge\ sem(S,s0,SR)\\ \ \wedge\ SR(\texttt{exc}) = \texttt{null}\end{array}\right\} s \left\{\begin{array}{l}\exists SQ : sem(S,s,SQ)\\ \ \wedge\ \pi(SQ,s)\end{array}\right\}}\ [\text{weak}]$$

case b:

Semantic property yields:

$$sem(S,s0,SQ)\ \wedge\ SQ(\texttt{exc}) \neq \texttt{null}\ \wedge\ \tau(SQ(\texttt{exc})) \not\preceq T\ \ \Rightarrow\ \ sem(S,s,SQ)$$

$$\frac{\left\{\ \pi(S,s)\ \wedge\ sem(S,s0,SR)\ \right\} s0 \left\{\ \pi(SR,s)\ \right\}}{\left\{\begin{array}{l}\pi(S,s)\ \wedge\ sem(S,s0,SR)\\ \ \wedge\ SR(\texttt{exc}) \neq \texttt{null}\\ \ \wedge\ \tau(SR(\texttt{exc})) \not\preceq T\end{array}\right\} s0 \left\{\begin{array}{l}sem(S,s0,SR)\\ \ \wedge\ SR(\texttt{exc}) \neq \texttt{null}\\ \ \wedge\ \tau(SR(\texttt{exc})) \not\preceq T\ \wedge\ \pi(SR,s)\end{array}\right\}}\ [\text{inva}]$$

$$\frac{\left\{\begin{array}{l}\pi(S,s)\\ \ \wedge\ sem(S,s0,SR)\\ \ \wedge\ SR(\texttt{exc}) \neq \texttt{null}\ \wedge\ \tau(SR(\texttt{exc})) \not\preceq T\end{array}\right\} s0 \left\{\begin{array}{l}sem(S,s,SR)\ \wedge\ \pi(SR,s)\\ \ \wedge\ \texttt{exc} \neq \texttt{null}\\ \ \wedge\ \tau(\texttt{exc}) \not\preceq T\end{array}\right\}}{\left\{\begin{array}{l}\pi(S,s)\ \wedge\ sem(S,s0,SR)\\ \ \wedge\ SR(\texttt{exc}) \neq \texttt{null}\ \wedge\ \tau(SR(\texttt{exc})) \not\preceq T\end{array}\right\} s \left\{\begin{array}{l}sem(S,s,SR)\\ \ \wedge\ \pi(SR,s)\end{array}\right\}}\ [\text{semp,weak}]\ [\text{catch-exc}]$$

$$\frac{}{\left\{\begin{array}{l}\pi(S,s)\ \wedge\ sem(S,s0,SR)\\ \ \wedge\ SR(\texttt{exc})\neq\texttt{null}\ \wedge\ \tau(SR(\texttt{exc}))\not\preceq T\end{array}\right\}s\left\{\begin{array}{l}\exists SQ: sem(S,s,SQ)\\ \ \wedge\ \pi(SQ,s)\end{array}\right\}}\ [\text{weak}]$$

case c:

Semantic property yields:

$$\begin{array}{l} sem(S,s0,SR)\ \wedge\ SR(\texttt{exc})\neq\texttt{null}\ \wedge\ \tau(SR(\texttt{exc}))\preceq T\\ \ \wedge\ ssem(SR[e:=SR(\texttt{exc}),\texttt{exc}:=\texttt{null}],s1,SQ)\\ \Rightarrow\\ sem(S,s,SQ)\end{array}$$

From the first induction hypo we get:

$$\left\{\ \pi(S,s)\ \wedge\ sem(S,s0,SR)\ \right\}s0\left\{\ \pi(SR,s)\ \right\}$$

$$\frac{}{\left\{\begin{array}{l}\pi(S,s)\ \wedge\ sem(S,s0,SR)\\ \ \wedge\ SR(\texttt{exc})\neq\texttt{null}\\ \ \wedge\ \tau(SR(\texttt{exc}))\preceq T\\ \ \wedge\ sem(SR[e:=SR(\texttt{exc}),\\ \qquad\qquad \texttt{exc}:=\texttt{null}],s1,SQ)\end{array}\right\}s0\left\{\begin{array}{l}SR(\texttt{exc})\neq\texttt{null}\\ \ \wedge\ \tau(SR(\texttt{exc}))\preceq T\\ \ \wedge\ sem(SR[e:=SR(\texttt{exc}),\\ \qquad\qquad \texttt{exc}:=\texttt{null}],s1,SQ)\\ \ \wedge\ \pi(SR,s)\end{array}\right\}}\ [\text{inva}]$$

$$\frac{}{\left\{\begin{array}{l}\pi(S,s)\ \wedge\ sem(S,s0,SR)\\ \ \wedge\ SR(\texttt{exc})\neq\texttt{null}\\ \ \wedge\ \tau(SR(\texttt{exc}))\preceq T\\ \ \wedge\ sem(SR[e:=SR(\texttt{exc}),\\ \qquad\qquad \texttt{exc}:=\texttt{null}],s1,SQ)\end{array}\right\}s0\left\{\begin{array}{l}sem(SR[e:=SR(\texttt{exc}),\\ \qquad\qquad \texttt{exc}:=\texttt{null}],s1,SQ)\\ \ \wedge\ \pi(SR,s)\ \wedge\ \texttt{exc}\neq\texttt{null}\\ \ \wedge\ \tau(\texttt{exc})\preceq T\end{array}\right\}}\ [\text{weak}]$$

$$\frac{}{\left\{\begin{array}{l}\pi(S,s)\ \wedge\ sem(S,s0,SR)\\ \ \wedge\ SR(\texttt{exc})\neq\texttt{null}\\ \ \wedge\ \tau(SR(\texttt{exc}))\preceq T\\ \ \wedge\ sem(SR[e:=SR(\texttt{exc}),\\ \qquad\qquad \texttt{exc}:=\texttt{null}],s1,SQ)\end{array}\right\}s0\left\{\begin{array}{l}sem(SR[e:=SR(\texttt{exc}),\\ \qquad\qquad \texttt{exc}:=\texttt{null}],s1,SQ)\\ \ \wedge\ \bigwedge_{v\in vis(s)\{\texttt{exc}\}}SR(v)=v\\ \ \wedge\ SR(\$)=\$\ \wedge\ SR(\texttt{exc})=\texttt{exc}\\ \ \wedge\ \texttt{exc}\neq\texttt{null}\ \wedge\ \tau(\texttt{exc})\preceq T\end{array}\right\}}\ [\text{weak}]$$

From the second induction hypo we get:

$$\left\{\begin{array}{l}sem(SR[e:=SR(\texttt{exc}),\texttt{exc}:=\texttt{null}],s1,SQ)\\ \ \wedge\ \pi(SR[e:=SR(\texttt{exc}),\texttt{exc}:=\texttt{null}],s1)\end{array}\right\}s1\left\{\ \pi(SQ,s1)\ \right\}$$

$$\frac{}{\left\{\begin{array}{l}sem(SR[e:=SR(\texttt{exc}),\texttt{exc}:=\texttt{null}],s1,SQ)\\ \ \wedge\ \bigwedge_{v\in vis(s1)\{\texttt{exc},e\}}SR(v)=v\ \wedge\ SR(\$)=\$\\ \ \wedge\ SR[e:=SR(\texttt{exc}),\texttt{exc}:=\texttt{null}](e)=e\\ \ \wedge\ \texttt{null}=\texttt{exc}\end{array}\right\}s1\left\{\ \pi(SQ,s)\ \right\}}\ [\text{pi}]$$

$$\frac{}{\left\{\begin{array}{l}sem(SR[e:=SR(\texttt{exc}),\texttt{exc}:=\texttt{null}],s1,SQ)\\ \ \wedge\ \bigwedge_{v\in vis(s1)\{\texttt{exc},e\}}SR(v)=v\\ \ \wedge\ SR(\$)=\$\ \wedge\ SR(\texttt{exc})=e\end{array}\right\}s1\left\{\ \pi(SQ,s)\ \right\}}\ [\text{swis}]$$

Applying the catching rule to the last two results yields

$$\left\{\begin{array}{l}\pi(S,s)\ \wedge\ sem(S,s0,SR)\ \wedge\ SR(\texttt{exc})\neq\texttt{null}\\ \ \wedge\ \tau(SR(\texttt{exc}))\preceq T\\ \ \wedge\ sem(SR[e:=SR(\texttt{exc}),\texttt{exc}:=\texttt{null}],s1,SQ)\end{array}\right\}s\left\{\ \pi(SQ,s)\ \right\}$$

$$\frac{}{\left\{\begin{array}{l}\pi(S,s)\ \wedge\ sem(S,s0,SR)\\ \ \wedge\ SR(\texttt{exc})\neq\texttt{null}\\ \ \wedge\ \tau(SR(\texttt{exc}))\preceq T\\ \ \wedge\ sem(SR[e:=SR(\texttt{exc}),\\ \qquad\qquad \texttt{exc}:=\texttt{null}],s1,SQ)\end{array}\right\}\ s\ \left\{\begin{array}{l}SR(\texttt{exc})\neq\texttt{null}\ \wedge\ sem(S,s0,SR)\\ \ \wedge\ \tau(SR(\texttt{exc}))\preceq T\\ \ \wedge\ sem(SR[e:=SR(\texttt{exc}),\\ \qquad\qquad \texttt{exc}:=\texttt{null}],s1,SQ)\\ \ \wedge\ \pi(SQ,s)\end{array}\right\}}\ [\text{inva}]$$

$$\frac{}{\left\{\begin{array}{l}\pi(S,s)\ \wedge\ sem(S,s0,SR)\\ \ \wedge\ SR(\texttt{exc})\neq\texttt{null}\\ \ \wedge\ \tau(SR(\texttt{exc}))\preceq T\\ \ \wedge\ sem(SR[e:=SR(\texttt{exc}),\\ \qquad\qquad \texttt{exc}:=\texttt{null}],s1,SQ)\end{array}\right\}\ s\ \left\{\begin{array}{l}\exists SQ: sem(S,s,SQ)\\ \ \wedge\ \pi(SQ,s)\end{array}\right\}}\ [\text{weak,semp}]$$

$$\frac{}{\left\{\begin{array}{l}\pi(S,s)\ \wedge\ sem(S,s0,SR)\\ \ \wedge\ SR(\texttt{exc})\neq\texttt{null}\ \wedge\ \tau(SR(\texttt{exc}))\preceq T\\ \ \wedge\ \exists SQQ:\\ \quad sem(SR[e:=SR(\texttt{exc}),\\ \qquad\qquad \texttt{exc}:=\texttt{null}],s1,SQQ)\end{array}\right\}\ s\ \left\{\begin{array}{l}\exists SQ: sem(S,s,SQ)\\ \ \wedge\ \pi(SQ,s)\end{array}\right\}}\ [\text{ex}]$$

In a similar way we show:

$$\left\{\ \pi(S,s)\ \wedge\ sem(S,s0,SR)\ \right\}\ s0\ \left\{\ \pi(SR,s)\ \right\}$$

$$\frac{}{\begin{array}{l}\left\{\begin{array}{l}\pi(S,s)\ \wedge\ sem(S,s0,SR)\ \wedge\ SR(\texttt{exc})\neq\texttt{null}\ \wedge\ \tau(SR(\texttt{exc}))\preceq T\\ \ \wedge\ \neg\exists SQQ: sem(SR[e:=SR(\texttt{exc}),\texttt{exc}:=\texttt{null}],s1,SQQ)\end{array}\right\}\\ s0\\ \left\{\begin{array}{l}SR(\texttt{exc})\neq\texttt{null}\ \wedge\ \tau(SR(\texttt{exc}))\preceq T\\ \ \wedge\ \neg\exists SQQ: sem(SR[e:=SR(\texttt{exc}),\texttt{exc}:=\texttt{null}],s1,SQQ)\ \wedge\ \pi(SR,s)\end{array}\right\}\end{array}}\ [\text{inva}]$$

$$\frac{}{\begin{array}{l}\left\{\begin{array}{l}\pi(S,s)\ \wedge\ sem(S,s0,SR)\ \wedge\ SR(\texttt{exc})\neq\texttt{null}\ \wedge\ \tau(SR(\texttt{exc}))\preceq T\\ \ \wedge\ \neg\exists SQQ: sem(SR[e:=SR(\texttt{exc}),\texttt{exc}:=\texttt{null}],s1,SQQ)\end{array}\right\}\\ s0\\ \left\{\begin{array}{l}\pi(SR,s)\ \wedge\ \texttt{exc}\neq\texttt{null}\ \wedge\ \tau(\texttt{exc})\preceq T\\ \ \wedge\ \neg\exists SQQ: sem(SR[e:=SR(\texttt{exc}),\texttt{exc}:=\texttt{null}],s1,SQQ)\end{array}\right\}\end{array}}\ [\text{weak}]$$

$$\frac{}{\begin{array}{l}\left\{\begin{array}{l}\pi(S,s)\ \wedge\ sem(S,s0,SR)\ \wedge\ SR(\texttt{exc})\neq\texttt{null}\ \wedge\ \tau(SR(\texttt{exc}))\preceq T\\ \ \wedge\ \neg\exists SQQ: sem(SR[e:=SR(\texttt{exc}),\texttt{exc}:=\texttt{null}],s1,SQQ)\end{array}\right\}\\ s0\\ \left\{\begin{array}{l}\bigwedge_{v\in vis(s)\{\texttt{exc}\}} SR(v)=v\ \wedge\ SR(\texttt{exc})=\texttt{exc}\ \wedge\ SR(\$)=\$\\ \ \wedge\ \texttt{exc}\neq\texttt{null}\ \wedge\ \tau(\texttt{exc})\preceq T\\ \ \wedge\ \neg\exists SQQ: sem(SR[e:=SR(\texttt{exc}),\texttt{exc}:=\texttt{null}],s1,SQQ)\end{array}\right\}\\ \left\{\begin{array}{l}\pi(SR[e:=SR(\texttt{exc}),\texttt{exc}:=\texttt{null}],s1)\\ \ \wedge\ \neg\exists SQQ: sem(SR[e:=SR(\texttt{exc}),\texttt{exc}:=\texttt{null}],s1,SQQ)\end{array}\right\}\ s1\ \left\{\ false\ \right\}\end{array}}\ [\text{ind-hypo}_d]$$

$$\frac{}{\left\{\begin{array}{l}\bigwedge_{v\in vis(s)\{\texttt{exc},e\}} SR(v)=v\\ \ \wedge\ SR(\texttt{exc})=e\ \wedge\ \texttt{null}=\texttt{exc}\ \wedge\ SR(\$)=\$\\ \ \wedge\ \neg\exists SQQ: sem(SR[\texttt{exc}:=\texttt{null}],s1,SQQ)\end{array}\right\}\ s1\ \left\{\ false\ \right\}}\ [\text{swis}]$$

$$\frac{}{\left\{\begin{array}{l}\bigwedge_{v\in vis(s)\{\texttt{exc},e\}} SR(v)=v\\ \ \wedge\ SR(\texttt{exc})=e\ \wedge\ SR(\$)=\$\\ \ \wedge\ \neg\exists SQQ: sem(SR[\texttt{exc}:=\texttt{null}],s1,SQQ)\end{array}\right\}\ s1\ \left\{\ false\ \right\}}\ [\text{swis}]$$

Applying the catching rule to the last two results yields

$$\frac{\left\{\begin{array}{l}\pi(S,s)\ \wedge\ sem(S,s0,SR)\\ \quad\wedge\ SR(\texttt{exc})\neq\texttt{null}\ \wedge\ \tau(SR(\texttt{exc}))\preceq T\\ \quad\wedge\ \neg\exists SQQ : sem(SR[e:=SR(\texttt{exc}),\texttt{exc}:=\texttt{null}],s1,SQQ)\end{array}\right\} s\left\{\ false\ \right\}}{\left\{\begin{array}{l}\pi(S,s)\ \wedge\ sem(S,s0,SR)\\ \quad\wedge\ SR(\texttt{exc})\neq\texttt{null}\ \wedge\ \tau(SR(\texttt{exc}))\preceq T\\ \quad\wedge\ \neg\exists SQQ : sem(SR[e:=SR(\texttt{exc}),\texttt{exc}:=\texttt{null}],\\ \qquad\qquad s1,SQQ)\end{array}\right\} s\left\{\begin{array}{l}\exists SQ : sem(S,s,SQ)\\ \quad\wedge\ \pi(SQ,s)\end{array}\right\}}\ [\text{weak}]$$

Disjunction leads to the goal for this case:

$$\left\{\begin{array}{l}\pi(S,s)\ \wedge\ sem(S,s0,SR)\\ \quad\wedge\ SR(\texttt{exc})\neq\texttt{null}\ \wedge\ \tau(SR(\texttt{exc}))\preceq T\end{array}\right\} s\left\{\begin{array}{l}\exists SQ : sem(S,s,SQ)\\ \quad\wedge\ \pi(SQ,s)\end{array}\right\}$$

QED catch

block(Type VarId Statement): $c == \{Tx; s\}$

Assuming

$$\mathcal{AVI}\quad\triangleright\quad\left\{\ \pi(S,s)\ \right\} s \left\{\ \exists SQ : sem(S,s,SQ)\ \wedge\ \pi(SQ,s)\ \right\}$$

we have to show

$$\mathcal{AVI}\quad\triangleright\quad\left\{\ \pi(S,c)\ \right\} c \left\{\ \exists SQ : sem(S,c,SQ)\ \wedge\ \pi(SQ,c)\ \right\}$$

Semantic property: $ssem(S[x:=\texttt{init}(T)],s,SQ) \Rightarrow ssem(S,c,SQ)$
and $wtp(S[x:=\texttt{init}(T)],s) \Rightarrow wtp(S,c)$ because $vis(V,s) \Leftrightarrow vis(V,c) \vee V = x'$
i.e. $sem(S[x:=\texttt{init}(T)],s,SQ)4 \Rightarrow sem(S,c,SQ)$

$$\frac{\left\{\ \pi(S[x:=\texttt{init}(T)],s)\ \right\} s \left\{\ \exists SQ : sem(S[x:=\texttt{init}(T)],s,SQ)\ \wedge\ \pi(SQ,s)\ \right\}}{\left\{\begin{array}{l}\pi(S[x:=\texttt{init}(T)],c)\\ \quad\wedge\ S[x:=\texttt{init}(T)](x)=x\end{array}\right\} s \left\{\begin{array}{l}\exists SQ : sem(S[x:=\texttt{init}(T)],s,SQ)\\ \quad\wedge\ \pi(SQ,s)\end{array}\right\}}\ [\text{strg}]$$

$$\frac{\left\{\ \pi(S[x:=\texttt{init}(T)],c)\ \right\} c \left\{\ \exists SQ : sem(S[x:=\texttt{init}(T)],s,SQ)\ \wedge\ \pi(SQ,s)\ \right\}}{\left\{\ \pi(S[x:=\texttt{init}(T)],c)\ \right\} c \left\{\ \exists SQ : sem(S[x:=\texttt{init}(T)],s,SQ)\ \wedge\ \pi(SQ,s)\ \right\}}\ [\text{block}]$$

$$\frac{\left\{\ \pi(S[x:=\texttt{init}(T)],c)\ \right\} c \left\{\ \exists SQ : sem(S,c,SQ)\ \wedge\ \pi(SQ,s)\ \right\}}{\left\{\ \pi(S[x:=\texttt{init}(T)],c)\ \right\} c \left\{\ \exists SQ : sem(S,c,SQ)\ \wedge\ \pi(SQ,s)\ \right\}}\ [\text{semp}]$$

$$\frac{}{\left\{\ \pi(S,c)\ \right\} c \left\{\ \exists SQ : sem(S,c,SQ)\ \wedge\ \pi(SQ,s)\ \right\}}\ [\text{strg, S[...] not applied to x}]$$

invoc(VarId VarId MethodId Exp): $c == x = y.m(e);$

We have to show

$$\mathcal{AVI}\quad\triangleright\quad\left\{\ \pi(S,c)\ \right\} x = y.m(e); \left\{\ \exists SQ : sem(S,c,SQ)\ \wedge\ \pi(SQ,c)\ \right\}$$

$\mathcal{AVI}$contains the following triple:

$$\left\{\ \pi(S,mv)\ \right\} mv \left\{\ \exists SQ : sem(S,mv,SQ)\ \wedge\ \rho(SQ,mv)\ \right\}$$

where $mv == vm(styp(y',c),m)$. The semantic property:

$$\begin{array}{l}S(y)\neq\texttt{null}\ \wedge\ \tau(S(y))\preceq styp(y',c)\\ \quad\wedge\ ssem(S[\texttt{this}:=S(y),\texttt{par}:=e[S],\\ \qquad\qquad\texttt{res}:=\texttt{init}(rtyp(dm(\tau(S(y)),m)))],\\ \qquad\qquad body(dm(\tau(S(y)),m)),SQ)\\ \Rightarrow\\ \quad ssem(S,c,S[x:=SQ(\texttt{res}),\$:=SQ(\$),\texttt{exc}:=SQ(\texttt{exc})]\end{array}$$

Omitting the assumptions, we obtain by assumpt-axiom and assumpt-intro:

$$\frac{\{\ \pi(S,mv)\ \}\,mv\,\{\ \exists SQ : sem(S,mv,SQ)\ \wedge\ \rho(SQ,mv)\ \}}{\{\ \pi(S,mv)\ \}\,mv\,\{\ \exists SQ : sem(S,dm(\tau(S(\texttt{this})),m),SQ))\ \wedge\ \rho(SQ,mv)\ \}}\ \text{[semd,weak]}$$

$$\frac{\{\ \pi(S,mv)\ \}\,mv\,\{\ \exists SQ : sem(S,dm(\tau(S(\texttt{this})),m),SQ))\ \wedge\ \rho(SQ,mv)\ \}}{\begin{array}{l}\{\ \pi(S[\texttt{this} := S(y), \texttt{par} := e[S]], mv)\ \}\\ mv\\ \left\{\begin{array}{l}\exists SQ : sem(S[\texttt{this} := S(y), \texttt{par} := e[S]],\\ \qquad dm(\tau(S[\texttt{this} := S(y), \texttt{par} := e[S]](\texttt{this})),m),SQ)\\ \ \wedge\ \rho(SQ,mv)\end{array}\right\}\end{array}}\ \begin{array}{l}[\ \text{swis},\\ S[\texttt{this} := S(y),\\ \texttt{par} := e[S]]forS\ ]\end{array}$$

$$\frac{\{\ \pi(S[\texttt{this} := S(y), \texttt{par} := e[S]], mv)\ \}\ mv\ \left\{\begin{array}{l}\exists SQ : sem(S[\texttt{this} := S(y), \texttt{par} := e[S]],\\ \qquad dm(\tau(S[\texttt{this} := S(y), \texttt{par} := e[S]](\texttt{this})),m),SQ)\\ \ \wedge\ \rho(SQ,mv)\end{array}\right\}}{\{\ \pi(S[\texttt{this} := S(y), \texttt{par} := e[S]], mv)\ \}\ mv\ \{\ \exists SQ : sem(S[\texttt{this} := S(y), \texttt{par} := e[S]], dm(\tau(S(y)),m),SQ)\ \wedge\ \rho(SQ,mv)\ \}}\ \text{[weak]}$$

$$\frac{\{\ \pi(S[\texttt{this} := S(y), \texttt{par} := e[S]], mv)\ \}\ mv\ \{\ \exists SQ : sem(S[\texttt{this} := S(y), \texttt{par} := e[S]], dm(\tau(S(y)),m),SQ)\ \wedge\ \rho(SQ,mv)\ \}}{\begin{array}{l}\{\ y \neq \texttt{null}\ \wedge\ \pi(S[\texttt{this} := S(y), \texttt{par} := e[S]], mv)[y/\texttt{this}, e/\texttt{par}]\ \}\\ x = y.m(e);\\ \left\{\begin{array}{l}(\exists SQ : sem(S[\texttt{this} := S(y), \texttt{par} := e[S]], dm(\tau(S(y)),m),SQ)\\ \ \wedge\ \rho(SQ,mv))[x/\texttt{res}]\end{array}\right\}\end{array}}\ \text{[invoc]}$$

$$\frac{\begin{array}{l}\{\ y \neq \texttt{null}\ \wedge\ \pi(S[\texttt{this} := S(y), \texttt{par} := e[S]], mv)[y/\texttt{this}, e/\texttt{par}]\ \}\\ x = y.m(e);\\ \left\{\begin{array}{l}(\exists SQ : sem(S[\texttt{this} := S(y), \texttt{par} := e[S]], dm(\tau(S(y)),m),SQ)\\ \ \wedge\ \rho(SQ,mv))[x/\texttt{res}]\end{array}\right\}\end{array}}{\begin{array}{l}\left\{\begin{array}{l}y \neq \texttt{null}\ \wedge\ (S[\texttt{this} := S(y), \texttt{par} := e[S]](\texttt{this}) = \texttt{this}\\ \qquad\qquad \wedge\ S[\texttt{this} := S(y), \texttt{par} := e[S]](\texttt{par}) = \texttt{par}\\ \qquad\qquad \wedge\ S[\texttt{this} := S(y), \texttt{par} := e[S]](\texttt{exc}) = \texttt{exc}\\ \qquad\qquad \wedge\ S[\texttt{this} := S(y), \texttt{par} := e[S]](\$) = \$)[y/\texttt{this}, e/\texttt{par}]\end{array}\right\}\\ x = y.m(e);\\ \left\{\begin{array}{l}(\exists SQ : sem(S[\texttt{this} := S(y), \texttt{par} := e[S]], dm(\tau(S(y)),m),SQ)\\ \ \wedge\ SQ(\texttt{res}) = \texttt{res}\ \wedge\ SQ(\texttt{exc}) = \texttt{exc}\ \wedge\ SQ(\$) = \$[x/\texttt{res}]\end{array}\right\}\end{array}}\ \text{[pi,rho]}$$

$$\frac{\begin{array}{l}\{\ y \neq \texttt{null}\ \wedge\ S(y) = y\ \wedge\ e[S] = e\ \wedge\ S(\texttt{exc}) = \texttt{exc}\ \wedge\ S(\$) = \$\ \}\\ x = y.m(e);\\ \left\{\begin{array}{l}\exists SQ : sem(S[\texttt{this} := S(y), \texttt{par} := e[S]], dm(\tau(S(y)),m),SQ)\\ \ \wedge\ SQ(\texttt{res}) = x\ \wedge\ SQ(\texttt{exc}) = \texttt{exc}\ \wedge\ SQ(\$) = \$\end{array}\right\}\end{array}}{\begin{array}{l}\left\{\begin{array}{l}y \neq \texttt{null}\ \wedge\ (S[\texttt{this} := S(y), \texttt{par} := e[S]](\texttt{this}) = \texttt{this}\\ \qquad\qquad \wedge\ S[\texttt{this} := S(y), \texttt{par} := e[S]](\texttt{par}) = \texttt{par}\\ \qquad\qquad \wedge\ S[\texttt{this} := S(y), \texttt{par} := e[S]](\texttt{exc}) = \texttt{exc}\\ \qquad\qquad \wedge\ S[\texttt{this} := S(y), \texttt{par} := e[S]](\$) = \$)[y/\texttt{this}, e/\texttt{par}]\end{array}\right\}\\ x = y.m(e);\\ \left\{\begin{array}{l}(\exists SQ : sem(S[\texttt{this} := S(y), \texttt{par} := e[S]], dm(\tau(S(y)),m),SQ)\\ \ \wedge\ SQ(\texttt{res}) = \texttt{res}\ \wedge\ SQ(\texttt{exc}) = \texttt{exc}\ \wedge\ SQ(\$) = \$[x/\texttt{res}]\end{array}\right\}\end{array}}\ \text{[swis]}$$

$$\frac{\begin{array}{l}\{\ y \neq \texttt{null}\ \wedge\ S(y) = y\ \wedge\ e[S] = e\ \wedge\ S(\texttt{exc}) = \texttt{exc}\ \wedge\ S(\$) = \$\ \}\\ x = y.m(e);\\ \left\{\begin{array}{l}\exists SQ : sem(S[\texttt{this} := S(y), \texttt{par} := e[S]], dm(\tau(S(y)),m),SQ)\\ \ \wedge\ SQ(\texttt{res}) = x\ \wedge\ SQ(\texttt{exc}) = \texttt{exc}\ \wedge\ SQ(\$) = \$\end{array}\right\}\end{array}}{\begin{array}{l}\{\ y \neq \texttt{null}\ \wedge\ S(y) = y\ \wedge\ e[S] = e\ \wedge\ S(\texttt{exc}) = \texttt{exc}\ \wedge\ S(\$) = \$\ \}\\ x = y.m(e);\\ \left\{\begin{array}{l}\exists SQ : sem(S[\texttt{this} := S(y), \texttt{par} := e[S], \texttt{res} := \texttt{init}(rtyp(dm(\tau(S(y)),m)))],\\ body(dm(\tau(S(y)),m)),SQ)\\ \ \wedge\ SQ(\texttt{res}) = x\ \wedge\ SQ(\texttt{exc}) = \texttt{exc}\ \wedge\ SQ(\$) = \$\end{array}\right\}\end{array}}\ \text{[semd]}$$

$$\frac{\begin{array}{l}\{\ y \neq \texttt{null}\ \wedge\ S(y) = y\ \wedge\ e[S] = e\ \wedge\ S(\texttt{exc}) = \texttt{exc}\ \wedge\ S(\$) = \$\ \}\\ x = y.m(e);\\ \left\{\begin{array}{l}\exists SQ : sem(S[\texttt{this} := S(y), \texttt{par} := e[S], \texttt{res} := \texttt{init}(rtyp(dm(\tau(S(y)),m)))],\\ body(dm(\tau(S(y)),m)),SQ)\\ \ \wedge\ SQ(\texttt{res}) = x\ \wedge\ SQ(\texttt{exc}) = \texttt{exc}\ \wedge\ SQ(\$) = \$\end{array}\right\}\end{array}}{\begin{array}{l}\{\ y \neq \texttt{null}\ \wedge\ S(y) = y\ \wedge\ e[S] = e\ \wedge\ S(\texttt{exc}) = \texttt{exc}\ \wedge\ S(\$) = \$\ \}\\ x = y.m(e);\\ \left\{\begin{array}{l}\exists SQ : S(y) \neq \texttt{null}\ \wedge\ \tau(S(y)) \preceq styp(y',c)\\ \ \wedge\ ssem(S[\texttt{this} := S(y), \texttt{par} := e[S], \texttt{res} := \texttt{init}(rtyp(dm(\tau(S(y)),m)))],\\ body(dm(\tau(S(y)),m)),SQ)\\ \ \wedge\ SQ(\texttt{res}) = x\ \wedge\ SQ(\texttt{exc}) = \texttt{exc}\ \wedge\ SQ(\$) = \$\end{array}\right\}\end{array}}\ \text{[inva,swis]}$$

$$\frac{\begin{array}{l}\{\ y \neq \texttt{null}\ \wedge\ S(y) = y\ \wedge\ e[S] = e\ \wedge\ S(\texttt{exc}) = \texttt{exc}\ \wedge\ S(\$) = \$\ \}\\ x = y.m(e);\\ \left\{\begin{array}{l}\exists SQ : S(y) \neq \texttt{null}\ \wedge\ \tau(S(y)) \preceq styp(y',c)\\ \ \wedge\ ssem(S[\texttt{this} := S(y), \texttt{par} := e[S], \texttt{res} := \texttt{init}(rtyp(dm(\tau(S(y)),m)))],\\ body(dm(\tau(S(y)),m)),SQ)\\ \ \wedge\ SQ(\texttt{res}) = x\ \wedge\ SQ(\texttt{exc}) = \texttt{exc}\ \wedge\ SQ(\$) = \$\end{array}\right\}\end{array}}{\begin{array}{l}\{\ y \neq \texttt{null}\ \wedge\ S(y) = y\ \wedge\ e[S] = e\ \wedge\ S(\texttt{exc}) = \texttt{exc}\ \wedge\ S(\$) = \$\ \}\\ x = y.m(e);\\ \left\{\begin{array}{l}\exists SQ : ssem(S,c,S[x := SQ(\texttt{res}), \$:= SQ(\$), \texttt{exc} := SQ(\texttt{exc})])\\ \ \wedge\ SQ(\texttt{res}) = x\ \wedge\ SQ(\texttt{exc}) = \texttt{exc}\ \wedge\ SQ(\$) = \$\end{array}\right\}\end{array}}\ \text{[semp,weak]}$$

$$\frac{}{\left\{\ y \neq \texttt{null}\ \wedge\ S(y) = y\ \wedge\ e[S] = e\ \wedge\ S(\texttt{exc}) = \texttt{exc}\ \wedge\ S(\$) = \$\ \wedge\ wtp(S,c)\ \right\}\ x = y.m(e);\ \left\{\begin{array}{l} \exists SQ : \exists SR : SR = S[x := SQ(\texttt{res}), \$:= SQ(\$), \texttt{exc} := SQ(\texttt{exc})] \\ \quad \wedge\ ssem(S,c,SR)\ \wedge\ SR(x) = x\ \wedge\ SR(\texttt{exc}) = \texttt{exc}\ \wedge\ SR(\$) = \$ \end{array}\right\}}\ [\text{weak}]$$

$$\frac{}{\left\{\begin{array}{l} y \neq \texttt{null}\ \wedge\ \bigwedge_{v \in vis(c)\{\texttt{exc},x\}} S(v) = V_v\ \wedge\ S(y) = y \\ \quad \wedge\ e[S] = e\ \wedge\ S(\texttt{exc}) = \texttt{exc}\ \wedge\ S(\$) = \$ \end{array}\right\}\ x = y.m(e);\ \left\{\begin{array}{l} \exists SQ : \exists SR : SR = S[x := SQ(\texttt{res}), \$:= SQ(\$), \texttt{exc} := SQ(\texttt{exc})] \\ \quad \wedge\ ssem(S,c,SR)\ \wedge\ \bigwedge_{v \in vis(c)\{\texttt{exc},x\}} S(v) = V_v\ \wedge\ SR(x) = x \\ \quad \wedge\ SR(\texttt{exc}) = \texttt{exc}\ \wedge\ SR(\$) = \$ \end{array}\right\}}\ [\text{inva}]$$

$$\frac{}{\left\{\begin{array}{l} y \neq \texttt{null}\ \wedge\ \bigwedge_{v \in vis(c)\{\texttt{exc},x\}} S(v) = v\ \wedge\ S(y) = y \\ \quad \wedge\ e[S] = e\ \wedge\ S(\texttt{exc}) = \texttt{exc}\ \wedge\ S(\$) = \$ \end{array}\right\}\ x = y.m(e);\ \left\{\begin{array}{l} \exists SQ : \exists SR : SR = S[x := SQ(\texttt{res}), \$:= SQ(\$), \texttt{exc} := SQ(\texttt{exc})] \\ \quad \wedge\ ssem(S,c,SR)\ \wedge\ \bigwedge_{v \in vis(c)\{\texttt{exc},x\}} S(v) = v\ \wedge\ SR(x) = x \\ \quad \wedge\ SR(\texttt{exc}) = \texttt{exc}\ \wedge\ SR(\$) = \$ \end{array}\right\}}\ [\text{invoc-var,n-times}]$$

$$\frac{}{\left\{\begin{array}{l} y \neq \texttt{null} \\ \quad \wedge\ \bigwedge_{v \in vis(c)\{\texttt{exc},x\}} S(v) = v \\ \quad \wedge\ S(y) = y\ \wedge\ e[S] = e \\ \quad \wedge\ S(\texttt{exc}) = \texttt{exc} \\ \quad \wedge\ S(\$) = \$ \end{array}\right\}\ x = y.m(e);\ \left\{\begin{array}{l} \exists SR : ssem(S,c,SR) \\ \quad \wedge\ \bigwedge_{v \in vis(c)\{\texttt{exc},x\}} SR(v) = v \\ \quad \wedge\ SR(x) = x \\ \quad \wedge\ SR(\texttt{exc}) = \texttt{exc} \\ \quad \wedge\ SR(\$) = \$ \end{array}\right\}}\ [\text{weak}]$$

$$\frac{}{\left\{\begin{array}{l} y \neq \texttt{null} \\ \quad \wedge\ \bigwedge_{v \in vis(c)} S(v) = v \\ \quad \wedge\ S(\$) = \$ \end{array}\right\}\ x = y.m(e);\ \left\{\begin{array}{l} \exists SR : ssem(S,c,SR) \\ \quad \wedge\ \bigwedge_{v \in vis(c)} SR(v) = v \\ \quad \wedge\ SR(\$) = \$ \end{array}\right\}}\ [\text{lm.ev,strg}]$$

$$\frac{}{\left\{\begin{array}{l} y \neq \texttt{null} \\ \quad \wedge\ \pi(S,c) \\ \quad \wedge\ wtp(S,c) \end{array}\right\}\ x = y.m(e);\ \left\{\begin{array}{l} \exists SR : wtp(S,c)\ \wedge\ ssem(S,c,SR) \\ \quad \wedge\ \bigwedge_{v \in vis(c)} SR(v) = v\ \wedge\ SR(\$) = \$ \end{array}\right\}}\ [\text{inva}]$$

$$\frac{}{\left\{\ y \neq \texttt{null}\ \wedge\ \pi(S,c)\ \right\}\ x = y.m(e);\ \left\{\ \exists SR : sem(S,c,SR)\ \wedge\ \pi(SR,c)\ \right\}}\ [\text{lemma.wtp,semd}]$$

Using invoc-axiom on the *mgf*-postcondition, and the semantic property:

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

we obtain the following proof outline:

$$\begin{array}{ll} & \{y = \texttt{null}\ \wedge\ \pi(S,c)\ \wedge\ wtp(S,c)\} \\ \Rightarrow & [[semd, swis]] \\ & \{y = \texttt{null}\ \wedge\ \pi(S,c)\ \wedge\ wtp(S,c)\ \wedge\ \exists SR : ssem(S,c,SR) \\ & \quad \wedge\ SR = S[\$:= S(\$)\langle \texttt{NullPExc}\rangle, \texttt{exc} := new(S(\$), \texttt{NullPExc})]\} \\ \Rightarrow & \\ & \{y = \texttt{null}\ \wedge\ \pi(S,c)\ \wedge\ \exists SR : sem(S,c,SR)\ \wedge\ \bigwedge_{v \in vis(c)\ \{\texttt{exc}\}} SR(v) = S(v) \\ & \quad \wedge\ SR(\texttt{exc}) = new(S(\$), \texttt{NullPExc})\ \wedge\ SR(\$) = S(\$)\langle \texttt{NullPExc}\rangle\} \\ \Rightarrow & [[strg, PL]] \\ & \left\{\begin{array}{l} y = \texttt{null}\ \wedge\ \exists SR : sem(S,c,SR) \\ \quad \wedge\ \bigwedge_{v \in vis(c)\ \{\texttt{exc}\}} SR(v) = v \\ \quad \wedge\ SR(\texttt{exc}) = new(\$, \texttt{NullPExc}) \\ \quad \wedge\ SR(\$) = \$\langle \texttt{NullPExc}\rangle \end{array}\right\}\ x = y.m(e);\ \left\{\begin{array}{l} \exists SR : sem(S,c,SR) \\ \quad \wedge\ \pi(SR,c) \end{array}\right\} \end{array}$$

Finally, application of the lemma.wtp and disjunction with the above result yields

$$\{\ \pi(S,c)\ \}\, x = y.m(e); \{\ \exists SR : sem(S,c,SR)\ \wedge\ \pi(SR,c)\ \}$$

QED invoc.

call(VarId CTypeId MethodId Exp): $c == x = super_T.m(e);$

We have to show

$$\mathcal{AVI} \quad \triangleright \quad \{\ \pi(S,c)\ \}\, x = super_T.m(e); \{\ \exists SQ : sem(S,c,SQ)\ \wedge\ \pi(SQ,c)\ \}$$

$\mathcal{AVI}$contains the following triple:

$$\{\ \pi(S,mT)\ \}\, mT \{\ \exists SQ : sem(S,mT,SQ)\ \wedge\ \rho(SQ,mT)\ \}$$

where $mT == dm(T,m)$.
Semantic property:

$$\begin{array}{l} ssem(S[\texttt{par} := e[S], \texttt{res} := \texttt{init}(rtyp(mT))], body(mT), SR) \\ \Rightarrow \\ ssem(S, c, S[x := SR(\texttt{res}), \$:= SR(\$), \texttt{exc} := SR(\texttt{exc})]) \end{array}$$

Omitting the assumptions, we obtain by assumpt-axiom and assumpt-intro:

$$\frac{\{\ \pi(S,mT)\ \}\, mT \{\ \exists SQ : sem(S,mT,SQ)\ \wedge\ \rho(SQ,mT)\ \}}{\{\ \pi(S[\texttt{par} := e[S]],mT)\ \}\, mT \left\{ \begin{array}{l} \exists SQ : sem(S[\texttt{par} := e[S]],mT,SQ) \\ \wedge\ \rho(SQ,mT) \end{array} \right\}}\ \text{[swis, S[par:=e[S]] for S]}$$

$$\frac{\{\ \pi(S[\texttt{par} := e[S]],mT)\ \}\, mT \left\{ \begin{array}{l} \exists SQ : sem(S[\texttt{par} := e[S]],mT,SQ) \\ \wedge\ \rho(SQ,mT) \end{array} \right\}}{\{\ \pi(S[\texttt{par} := e[S]],mT)\ \}\ mT \left\{ \begin{array}{l} \exists SQ : ssem(S[\texttt{par} := e[S], \texttt{res} := \texttt{init}(rtyp(mT))], body(mT), SQ) \\ \wedge\ \rho(SQ,mT) \end{array} \right\}}\ \text{[semd]}$$

$$\frac{\{\ \pi(S[\texttt{par} := e[S]],mT)\ \}\ mT \left\{ \begin{array}{l} \exists SQ : ssem(S[\texttt{par} := e[S], \texttt{res} := \texttt{init}(rtyp(mT))], body(mT), SQ) \\ \wedge\ \rho(SQ,mT) \end{array} \right\}}{\{\ \pi(S[\texttt{par} := e[S]],mT)\ \}\ mT \left\{ \begin{array}{l} \exists SQ : \exists SR : SR = S[x := SQ(\texttt{res}), \$:= SQ(\$), \texttt{exc} := SQ(\texttt{exc})] \\ \wedge\ ssem(S,c,SR)\ \wedge\ \rho(SQ,mT) \end{array} \right\}}\ \text{[semp]}$$

$$\frac{\{\ \pi(S[\texttt{par} := e[S]],mT)\ \}\ mT \left\{ \begin{array}{l} \exists SQ : \exists SR : SR = S[x := SQ(\texttt{res}), \$:= SQ(\$), \texttt{exc} := SQ(\texttt{exc})] \\ \wedge\ ssem(S,c,SR)\ \wedge\ \rho(SQ,mT) \end{array} \right\}}{\{\ \pi(S[\texttt{par} := e[S]],mT)[e/\texttt{par}]\ \}\ mT \left\{ \begin{array}{l} (\exists SQ : \exists SR : SR = S[x := SQ(\texttt{res}), \$:= SQ(\$), \texttt{exc} := SQ(\texttt{exc})] \\ \wedge\ ssem(S,c,SR)\ \wedge\ \rho(SQ,mT))[x/\texttt{res}] \end{array} \right\}}\ \text{[call]}$$

$$\frac{\{\ \pi(S[\texttt{par} := e[S]],mT)[e/\texttt{par}]\ \}\ mT \left\{ \begin{array}{l} (\exists SQ : \exists SR : SR = S[x := SQ(\texttt{res}), \$:= SQ(\$), \texttt{exc} := SQ(\texttt{exc})] \\ \wedge\ ssem(S,c,SR)\ \wedge\ \rho(SQ,mT))[x/\texttt{res}] \end{array} \right\}}{\{\ e[S] = e\ \wedge\ S(\texttt{this}) = \texttt{this}\ \wedge\ S(\texttt{exc}) = \texttt{exc}\ \wedge\ S(\$) = \$\ \}\ c \left\{ \begin{array}{l} \exists SQ : \exists SR : SR = S[x := SQ(\texttt{res}), \$:= SQ(\$), \texttt{exc} := SQ(\texttt{exc})] \\ \wedge\ ssem(S,c,SR)\ \wedge\ SQ(\texttt{res}) = x\ \wedge\ SQ(\texttt{exc}) = \texttt{exc}\ \wedge\ SQ(\$) = \$ \end{array} \right\}}\ \text{[weak]}$$

$$\frac{\{\ e[S] = e\ \wedge\ S(\texttt{this}) = \texttt{this}\ \wedge\ S(\texttt{exc}) = \texttt{exc}\ \wedge\ S(\$) = \$\ \}\ c \left\{ \begin{array}{l} \exists SQ : \exists SR : SR = S[x := SQ(\texttt{res}), \$:= SQ(\$), \texttt{exc} := SQ(\texttt{exc})] \\ \wedge\ ssem(S,c,SR)\ \wedge\ SQ(\texttt{res}) = x\ \wedge\ SQ(\texttt{exc}) = \texttt{exc}\ \wedge\ SQ(\$) = \$ \end{array} \right\}}{\left\{ \begin{array}{l} e[S] = e\ \wedge\ S(\texttt{this}) = \texttt{this} \\ \wedge\ S(\texttt{exc}) = \texttt{exc}\ \wedge\ S(\$) = \$ \end{array} \right\} c \left\{ \begin{array}{l} \exists SR : ssem(S,c,SR)\ \wedge\ SR(x) = x \\ \wedge\ SR(\texttt{exc}) = \texttt{exc}\ \wedge\ SR(\$) = \$ \end{array} \right\}}\ \text{[weak]}$$

$$\frac{\left\{ \begin{array}{l} e[S] = e\ \wedge\ S(\texttt{this}) = \texttt{this} \\ \wedge\ S(\texttt{exc}) = \texttt{exc}\ \wedge\ S(\$) = \$ \end{array} \right\} c \left\{ \begin{array}{l} \exists SR : ssem(S,c,SR)\ \wedge\ SR(x) = x \\ \wedge\ SR(\texttt{exc}) = \texttt{exc}\ \wedge\ SR(\$) = \$ \end{array} \right\}}{\{\ e[S] = e\ \wedge\ \pi(S,c)\ \}\, c \{\ \exists SR : ssem(S,c,SR)\ \wedge\ \pi(SR,c)\ \}}\ \text{[call-var]}$$

$$\frac{\{\ e[S] = e\ \wedge\ \pi(S,c)\ \}\, c \{\ \exists SR : ssem(S,c,SR)\ \wedge\ \pi(SR,c)\ \}}{\{\ \pi(S,c)\ \wedge\ wtp(S,c)\ \}\, c \{\ \exists SR : wtp(S,c)\ \wedge\ ssem(S,c,SR)\ \wedge\ \pi(SR,c)\ \}}\ \text{[lm.ev,inva]}$$

$$\text{[lm.wtp,semd]}$$

$\{\ \pi(S,c)\ \}\, x = super_T.m(e); \{\ \exists SQ : sem(S,c,SQ)\ \wedge\ \pi(SQ,c)\ \}$

QED call

AD 2:

Derive $\mathcal{AI} \models mgf(mv^i)$ for all mv^i by induction on the type hierarchy of M. (In the proofs, we keep $\mathcal{AI}$ implicit!)

Induction base:

Let $mv^i = T0 : m$ such that $T0$ has only `null`as subtype in M:

1. $T0$ is declared by an interface. I.e.:

   $\tau(\texttt{this}) \preceq T0 \Rightarrow \texttt{this} = \texttt{null}$

   We have to derive:

   $$\mathcal{AI} \models \{\ \pi(S, T0:m)\ \}\, T0:m \left\{ \begin{array}{l} \exists SR : sem(S, T0:m, SR) \\ \wedge\ \rho(SR, T0:m) \end{array} \right\}$$

   $$\frac{\begin{array}{l} \tau(\texttt{this}) \preceq T0\ \wedge\ \texttt{this} \neq \texttt{null}\ \wedge\ \pi(S,T0:m) \Rightarrow false \\ \{\ false\ \}\, T0:m\, \{\ false\ \} \\ false \Rightarrow \exists SR : sem(S,T0:m,SR)\ \wedge\ \rho(SR,T0:m) \end{array}}{\left\{ \begin{array}{l} \tau(\texttt{this}) \preceq T0 \\ \wedge\ \pi(S,T0:m) \end{array} \right\} T0:m \left\{ \begin{array}{l} \exists SR : sem(S,T0:m,SR) \\ \wedge\ \rho(SR,T0:m) \end{array} \right\}}\ [\text{swis}]$$

2. $T0$ is declared by a class: We have to derive:

   $$\mathcal{AI} \models \{\ \pi(S,T0:m)\ \}\, T0:m\, \{\ \exists SR : sem(S,T0:m,SR)\ \wedge\ \rho(SR,T0:m)\ \}$$

   according to the class-rule it suffices to show

   (a) $$\left\{ \begin{array}{l} \tau(\texttt{this}) = T0 \\ \wedge\ \pi(S,T0:m) \end{array} \right\} dm(T0,m) \left\{ \begin{array}{l} \exists SR : sem(S,T0:m,SR) \\ \wedge\ \rho(SR,T0:m) \end{array} \right\}$$

   (b) $$\left\{ \begin{array}{l} \tau(\texttt{this}) \prec T0 \\ \wedge\ \pi(S,T0:m) \end{array} \right\} T0:m \left\{ \begin{array}{l} \exists SR : sem(S,T0:m,SR) \\ \wedge\ \rho(SR,T0:m) \end{array} \right\}$$

   AD a:

   By assumpt-axiom and assumpt-intro, we get:
   $\mathcal{AI} \models \{\ \pi(S, dm(T0,m))\ \}\, dm(T0,m)\, \{\ \exists SR : sem(S, dm(T0,m), SR)\ \wedge\ \rho(SR, dm(T0,m))\ \}$

   $$\frac{\left\{ \begin{array}{l} \pi(S, dm(T0,m)) \end{array} \right\} dm(T0,m) \left\{ \begin{array}{l} \exists SR : sem(S, dm(T0,m), SR) \\ \wedge\ \rho(SR, dm(T0,m)) \end{array} \right\}}{\left\{ \begin{array}{l} \tau(S(\texttt{this})) = T0 \\ \wedge\ \pi(S, dm(T0,m)) \end{array} \right\} dm(T0,m) \left\{ \begin{array}{l} \tau(S(\texttt{this})) = T0 \\ \wedge\ \exists SR : sem(S, dm(T0,m), SR) \\ \wedge\ \rho(SR, dm(T0,m)) \end{array} \right\}}\ [\text{inva}]$$

   $$\frac{\left\{ \begin{array}{l} \tau(S(\texttt{this})) = T0 \\ \wedge\ \pi(S, dm(T0,m)) \end{array} \right\} dm(T0,m) \left\{ \begin{array}{l} \tau(S(\texttt{this})) = T0 \\ \wedge\ \exists SR : sem(S, dm(T0,m), SR) \\ \wedge\ \rho(SR, dm(T0,m)) \end{array} \right\}}{\left\{ \begin{array}{l} \tau(\texttt{this}) = T0\ \wedge\ \pi(S, dm(T0,m)) \end{array} \right\} dm(T0,m)}\ [\text{strg,semd}]$$

$$\left\{ \begin{array}{l} \tau(S(\texttt{this})) = T0 \\ \quad \wedge\ \exists SR : wtp(S, dm(T0, m)) \ \wedge\ sem(S, dm(T0, m), SR) \\ \qquad\qquad\qquad\qquad \wedge\ \rho(SR, dm(T0, m)) \end{array} \right\}$$

[swis]

$$\left\{ \begin{array}{l} \tau(\texttt{this}) = T0 \\ \quad \wedge\ \pi(S, T0 : m) \end{array} \right\} dm(T0, m) \left\{ \begin{array}{l} \tau(S(\texttt{this})) = T0 \\ \quad \wedge\ \exists SR : wtp(S, T0 : m) \\ \quad \wedge\ sem(S, dm(\tau(S(\texttt{this})), m), SR) \\ \quad \wedge\ \rho(SR, T0 : m)) \end{array} \right\}$$

[def *sem*]

$$\left\{ \ \tau(\texttt{this}) = T0 \ \wedge\ \pi(S, T0 : m) \ \right\} dm(T0, m) \left\{ \begin{array}{l} \exists SR : sem(S, T0 : m, SR) \\ \quad \wedge\ \rho(SR, T0 : m) \end{array} \right\}$$

AD b:

According to the case assumption we have $\tau(\texttt{this}) \prec T0 \Rightarrow \texttt{this} = \texttt{null}$, i.e.

$\tau(\texttt{this}) \prec T0 \ \wedge\ \texttt{this} \neq \texttt{null} \ \wedge\ \pi(S, T0 : m) \ \Rightarrow\ false \ \left\{ \ false \ \right\} T0 : m \left\{ \ false \ \right\} false \ \Rightarrow$
$\exists SR : sem(S, T0 : m, SR) \ \wedge\ \rho(SR, T0 : m)$

[swis]

$$\left\{ \ \tau(\texttt{this}) \prec T0 \ \wedge\ \pi(S, T0 : m) \ \right\} T0 : m \left\{ \ \exists SR : sem(S, T0 : m, SR) \ \wedge\ \rho(SR, T0 : m) \ \right\}$$

Induction step:

Let $mv^i = T0 : m$ such that $T0$ has the direct subtypes $T1, \ldots, Tn$ in M: $T \prec T0 \Leftrightarrow \bigvee_i T \preceq Ti$

The properties of the subtypes can be combined as follows:

$$\left\{ \ \pi(S, Ti : m) \ \right\} Ti : m \left\{ \ \exists SR : sem(S, Ti : m, SR) \ \wedge\ \rho(SR, Ti : m) \ \right\}$$

[semd]

$$\left\{ \ \pi(S, Ti : m) \ \right\} Ti : m \left\{ \begin{array}{l} \exists SR : wtp(S, Ti : m) \ \wedge \\ sem(S, dm(\tau(S(\texttt{this})), m), SR) \\ \quad \wedge\ \rho(SR, Ti : m) \end{array} \right\}$$

[S(this) $\preceq$ Ti $\prec$ T0]

$$\left\{ \ \pi(S, Ti : m) \ \right\} Ti : m \left\{ \begin{array}{l} \exists SR : wtp(S, T0 : m) \\ \quad \wedge\ sem(S, dm(\tau(S(\texttt{this})), m), SR) \\ \quad \wedge\ \rho(SR, Ti : m) \end{array} \right\}$$

[semd]

$$\left\{ \ \pi(S, Ti : m) \ \right\} Ti : m \left\{ \ \exists SR : sem(S, T0 : m, SR) \ \wedge\ \rho(SR, Ti : m) \ \right\}$$

[def pi,rho]

$$\left\{ \ \pi(S, T0 : m) \ \right\} Ti : m \left\{ \ \exists SR : sem(S, T0 : m, SR) \ \wedge\ \rho(SR, T0 : m) \ \right\}$$

[Ti $\preceq$ T0,subtype]

$$\left\{ \begin{array}{l} \tau(\texttt{this}) \preceq Ti \\ \quad \wedge\ \pi(S, T0 : m) \end{array} \right\} T0 : m \left\{ \begin{array}{l} \exists SR : sem(S, T0 : m, SR) \\ \quad \wedge\ \rho(SR, T0 : m) \end{array} \right\}$$

and by successive application of the disjunct rule:

$$\left\{ \begin{array}{l} \bigvee_i \tau(\texttt{this}) \preceq Ti \\ \quad \wedge\ \pi(S, T0 : m) \end{array} \right\} T0 : m \left\{ \begin{array}{l} \exists SR : sem(S, T0 : m, SR) \\ \quad \wedge\ \rho(SR, T0 : m) \end{array} \right\}$$

[swis,$T \prec T0 \Leftrightarrow \bigvee_i T \preceq Ti$]

$$\left\{ \begin{array}{l} \tau(\texttt{this}) \prec T0 \\ \quad \wedge\ \pi(S, T0 : m) \end{array} \right\} T0 : m \left\{ \begin{array}{l} \exists SR : sem(S, T0 : m, SR) \\ \quad \wedge\ \rho(SR, T0 : m) \end{array} \right\}$$

1. $T0$ is declared by an interface ($\tau(\texttt{this}) \neq T0$).We have to derive:

$$\mathcal{AI} \models \left\{\ \pi(S, T0:m)\ \right\} T0:m \left\{\ \exists SR : sem(S, T0:m, SR)\ \wedge\ \rho(SR, T0:m)\ \right\}$$

Using the combined properties of the subtypes, we get:

$$\frac{\left\{\begin{array}{l}\tau(\texttt{this}) \prec T0 \\ \ \wedge\ \pi(S, T0:m)\end{array}\right\} T0:m \left\{\begin{array}{l}\exists SR : sem(S, T0:m, SR) \\ \ \wedge\ \rho(SR, T0:m)\end{array}\right\}}{\left\{\begin{array}{l}\tau(\texttt{this}) \preceq T0 \\ \ \wedge\ \pi(S, T0:m)\end{array}\right\} T0:m \left\{\begin{array}{l}\exists SR : sem(S, T0:m, SR) \\ \ \wedge\ \rho(SR, T0:m)\end{array}\right\}}\ [\tau(\texttt{this}) \neq \text{T0,strg}]$$

$$\frac{\left\{\begin{array}{l}\tau(\texttt{this}) \preceq T0 \\ \ \wedge\ \pi(S, T0:m)\end{array}\right\} T0:m \left\{\begin{array}{l}\exists SR : sem(S, T0:m, SR) \\ \ \wedge\ \rho(SR, T0:m)\end{array}\right\}}{\left\{\ \pi(S, T0:m)\ \right\} T0:m \left\{\ \exists SR : sem(S, T0:m, SR)\ \wedge\ \rho(SR, T0:m)\ \right\}}\ [\text{swis}]$$

2. $T0$ is declared by a class: We have to derive:

$$\mathcal{AI} \models \left\{\ \pi(S, T0:m)\ \right\} T0:m \left\{\ \exists SR : sem(S, T0:m, SR)\ \wedge\ \rho(SR, T0:m)\ \right\}$$

This can be derived by the class rule from the combined properties of the subtypes and based on

$$\mathcal{AI} \models \left\{\begin{array}{l}\tau(\texttt{this}) = T0 \\ \ \wedge\ \pi(S, T0:m)\end{array}\right\} dm(T0, m) \left\{\begin{array}{l}\exists SR : sem(S, T0:m, SR) \\ \ \wedge\ \rho(SR, T0:m)\end{array}\right\}$$

which was already derived from $\mathcal{AI}$ in the induction base (2.a).

QED.ad2

AD 3:

Derive $\mathcal{AI} \models mgf(\text{c})$ for all statements by assumption elimination:

We have proved in (1.) $\mathcal{AVI} \models mgf(\text{c})$ for all statements and in (2.) $\mathcal{AI} \models mgf(\text{mv}^i)$ for mv^i. That is the assumptions of $\mathcal{AV}$ are proved under the assumption $\mathcal{AI}$ and can thus be eliminated step by step using assumpt-intro and assumpt-elim.

AD 4:

We may assume $\mathcal{AI} \models mgf(body(mi^j))$ from (3.). We start with $body(mi^1)$, abbreviated as $bd1$. Denoting $mgf(mi^2), \ldots, mgf(mi^r)$ by $\mathcal{AI}\backslash 1$, we have:

$$\begin{array}{lll}\mathcal{AI}\backslash 1, & \left\{\ \pi(S, mi^1)\ \right\} mi^1 \left\{\ \exists SR : sem(S, mi^1, SR)\ \wedge\ \rho(SR, mi^1)\ \right\} \\ \models & \left\{\ \pi(S, bd1))\ \right\} bd1 \left\{\ \exists SR : sem(S, bd1, SR)\ \wedge\ \rho(SR, body(mi^1))\ \right\}\end{array}$$

To be able to apply impl-rule, we adapt the triple:

$$\frac{\left\{\ \pi(S, bd1)\ \right\} bd1 \left\{\begin{array}{l}\exists SR : sem(S, bd1, SR) \\ \ \wedge\ \rho(SR, bd1)\end{array}\right\}}{\left\{\begin{array}{l}\pi(S[\texttt{res} := \texttt{init}(rtyp(mi^1))], \\ \ bd1)\end{array}\right\} bd1 \left\{\begin{array}{l}\exists SR : sem(S[\texttt{res} := \texttt{init}(rtyp(mi^1)], \\ \qquad bd1, SR) \\ \ \wedge\ \rho(SR, bd1)\end{array}\right\}}\ [\text{inst}]$$

$$\frac{\left\{\begin{array}{l}\pi(S[\texttt{res} := \texttt{init}(rtyp(mi^1))], \\ \ bd1)\end{array}\right\} bd1 \left\{\begin{array}{l}\exists SR : sem(S[\texttt{res} := \texttt{init}(rtyp(mi^1)], \\ \qquad bd1, SR) \\ \ \wedge\ \rho(SR, bd1)\end{array}\right\}}{\left\{\begin{array}{l}S(\texttt{this}) = \texttt{this} \\ \ \wedge\ S(\texttt{par}) = \texttt{par} \\ \ \wedge\ \texttt{init}(rtyp(mi^1)) = \texttt{res} \\ \ \wedge\ S(\texttt{exc}) = \texttt{exc} \\ \ \wedge\ S(\$) = \$\end{array}\right\} bd1 \left\{\begin{array}{l}\exists SR : sem(S[\texttt{res} := \texttt{init}(rtyp(mi^1)], \\ \qquad bd1, SR) \\ \ \wedge\ wtp(S[\texttt{res} := \texttt{init}(rtyp(mi^1)], bd1) \\ \ \wedge\ SR(\texttt{this}) = \texttt{this}\ \wedge\ SR(\texttt{par}) = \texttt{par} \\ \ \wedge\ SR(\texttt{exc}) = \texttt{exc}\ \wedge\ SR(\texttt{res}) = \texttt{res} \\ \ \wedge\ SR(\$) = \$\end{array}\right\}}\ [\text{pi,rho}]$$

$$\frac{\left\{\begin{array}{l} S(\texttt{this}) = \texttt{this} \\ \quad \wedge\ S(\texttt{par}) = \texttt{par} \\ \quad \wedge\ \texttt{init}(rtyp(mi^1)) = \texttt{res} \\ \quad \wedge\ S(\texttt{exc}) = \texttt{exc} \\ \quad \wedge\ S(\$) = \$ \end{array}\right\} bd1 \left\{\begin{array}{l} \exists SR : sem(S[\texttt{res} := \texttt{init}(rtyp(mi^1))], \\ \qquad\qquad bd1, SR) \\ \quad \wedge\ wtp(S, mi^1) \\ \quad \wedge\ SR(\texttt{this}) = \texttt{this} \ \wedge\ SR(\texttt{par}) = \texttt{par} \\ \quad \wedge\ SR(\texttt{exc}) = \texttt{exc} \ \wedge\ SR(\texttt{res}) = \texttt{res} \\ \quad \wedge\ SR(\$) = \$ \end{array}\right\}}{\left\{\begin{array}{l} \pi(S, mi^1) \\ \quad \wedge\ \texttt{init}(rtyp(mi^1)) = \texttt{res} \end{array}\right\} bd1 \left\{\begin{array}{l} \exists SR : sem(S, mi^1, SR) \\ \quad \wedge\ \rho(SR, mi^1) \end{array}\right\}} \begin{array}{l}\text{[weak,semd]} \\ \text{[weak,semd]}\end{array}$$

By impl-rule we conclude:

$$\mathcal{AI}\backslash 1 \mid\!\rhd\ mgf(mi^1)$$

By assumpt-elim rule, we show for all j:

$$\mathcal{AI}\backslash 1 \mid\!\rhd\ mgf(body(mi^j))$$

Using the same procedure, we obtain $mgf(mi^j + 1), \ldots, mgf(mi^r) \mid\!\rhd\ mgf(mi^j)$ for all mi^j.
QED.

AD 5:

We have

$$\begin{array}{rcl} mgf(mi^2), \ldots, mgf(mi^r) & \mid\!\rhd & mgf(mi^1) \\ mgf(mi^3), \ldots, mgf(mi^r) & \mid\!\rhd & mgf(mi^2) \\ & \vdots & \\ mgf(mi^r) & \mid\!\rhd & mgf(mi^{r-1}) \\ & \mid\!\rhd & mgf(mi^r) \end{array}$$

Using assumpt-intro and assumpt-elim the assumptions can be eliminated starting with $mgf(mi^r), mgf(mi^{r-1})$, etc. QED.

AD 6:

Derive $\mid\!\rhd\ mgf(mv^i)$ for all mv^i by assumption elimination:

We have proved in (2.): $\mathcal{AI} \mid\!\rhd\ mgf(mv^i)$ for all mv^i and in (5.) $\mid\!\rhd\ mgf(mi^j)$. That is all assumptions in $\mathcal{AI}$ are proved and can be eliminated step by step using assumpt-intro and assumpt-elim.

AD 7:

Derive $\mid\!\rhd\ mgf(c)$ for all statements by assumption elimination:

We have proved in (3.): $\mathcal{AI} \mid\!\rhd\ mgf(c)$ for all statements and in (5.) $\mid\!\rhd\ mgf(mi^j)$ for all mv^i. That is all assumptions in $\mathcal{AI}$ are proved and can be eliminated step by step using assumpt-intro and assumpt-elim.

References

[1] Joachim van den Berg and Bart Jacobs. The LOOP compiler for Java and JML. In T. Margaria and W. Yi, editors, *TACAS01, Tools and Algorithms for the Construction and Analysis of Systems*, volume 2031 of *Lecture Notes in Computer Science*, pages 299–312. Springer-Verlag, 2001.

[2] Judy Crow, Sam Owre, John Rushby, Natarajan Shankar, and Mandayam Srivas. A tutorial introduction to PVS. In *Workshop on Industrial-Strength Formal Specification Techniques*, Boca Raton, Florida, April 1995.

[3] T. Gergely and L. Úry. *First-Order Programming Theories*. Springer-Verlag, 1991.

[4] Thomas Kleymann. Hoare logic and auxiliary variables. *Formal Aspects of Computing*, 11(5):541–566, 1999. Extended version of Schreiber, Auxiliary Variables and Recursive Procedures.

[5] K. Rustan M. Leino. Ecstatic: An object-oriented programming language with an axiomatic semantics. In B. Pierce, editor, *Proceedings of the Fourth International Workshop on Foundations of Object-Oriented Languages*, 1997. Available from: `http://www.cis.upenn.edu/~bcpierce/fool/`.

[6] K. Rustan M. Leino, Greg Nelson, and James B. Saxe. ESC/Java's User's Manual. Technical Note 2000-002, Compaq Systems Research Center, October 2000. Available from `http://gatekeeper.dec.com/pub/DEC/SRC/technical-notes/SRC-2000-002.html`.

[7] Jörg Meyer and Arnd Poetzsch-Heffter. An architecture for interactive program provers. In S. Graf and M. Schwartzbach, editors, *TACAS00, Tools and Algorithms for the Construction and Analysis of Systems*, volume 1785 of *Lecture Notes in Computer Science*, pages 63–77. Springer-Verlag, 2000.

[8] Peter Müller. *Modular Specification and Verification of Object-Oriented Programs*, volume 2262 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[9] David von Oheimb. *Analyzing Java in Isabelle/HOL: Formalization, Type Safety and Hoare Logic*. PhD thesis, Technische Universität München, 2001. `http://www4.in.tum.de/~oheimb/diss/`.

[10] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, New York, NY, USA, 1994.

[11] Arnd Poetzsch-Heffter. Specification and verification of object-oriented programs. Habilitation thesis, Technical University of Munich, January 1997.

[12] Arnd Poetzsch-Heffter and Peter Müller. A programming logic for sequential Java. In S. Doaitse Swierstra, editor, *ESOP '99: Proceedings of the 8^{th} European Symposium on Programming Languages and Systems*, volume 1576 of *Lecture Notes in Computer Science*, pages 162–176. Springer-Verlag, 1999.

[13] Nicole Rauch and Burkhart Wolff. Formalizing Java's two's-complement integral type in Isabelle/HOL. In Thomas Arts and Wan Fokkink, editors, *Proc. 8th International Workshop on Formal Methods for Industrial Critical Systems*, volume 80 of *Electronic Notes in Theoretical Computer Science (ENTCS)*, pages 40–56, Røros, Norway, June 2003. Elsevier.