Compiling Proof Obligations

Mariela Pavlova

June 28, 2006

Contents

1 Introduction

This documents studies the relationship between the verification conditions generated for a Java like source language and the verification conditions generated for the bytecode language defined in [3]. We establish an equivalence which we name $=^{mod\ Names\ and\ bools}$ modulo names and boolean values of the proof obligations on source and bytecode level. This result may have an impact on the application on PCC techniques for complex functional and security properties where full automatisation is not possible.

The traditional PCC architecture comes along with a certifying compiler. The basic idea is that the certifying compiler infers automatically annotations, automatically generates verification conditions, proves them automatically and then sends both the code and the proof certificate to the counterpart that will run the code. The receiver then, generates the verification conditions and type checks the generated formulas against the proof certificate. This architecture works for properties like well typedness and safe memory read/write but it is not applicable for complex policies where the specification and the proof cannot be done automatically.

...

2 Source

We present a source Java-like programming language which supports the following features: object manipulation and creation, method invokation, throwing and handling exceptions, subroutines etc. The first definition that we give hereafter presents all the constructs of our language which evaluate to a value.

Definition 2.1 (Expression) *The grammar for source expressions is defined as follows*

$$
\begin{array}{rl}
\mathcal{E}^{src} ::= & \mathbf{constInt} \\
& \mid \mathbf{true} \\
& \mid \mathbf{false} \\
& \mid \mathcal{E}^{src}\ op\ \mathcal{E}^{src} \\
& \mid \mathcal{E}^{src}.f \\
& \mid \mathbf{var} \\
& \mid (Class)\ \mathcal{E}^{src} \\
& \mid \mathbf{null} \\
& \mid \mathbf{this} \\
& \mid \mathcal{E}^{srcRel} \\
& \mid \mathcal{E}^{src}.m(\mathcal{E}^{src}) \\
& \mid \mathbf{new}\ Class(\mathcal{E}^{src})
\end{array}
$$

$$
\begin{array}{rl}
\mathcal{E}^{srcRel} ::= & \mathcal{E}^{src}\ \mathcal{R}\ \mathcal{E}^{src} \\
& \mid \mathcal{E}^{src}\ \mathbf{instanceof}\ Class \\
& \\
& \mathcal{R} \in \{\leq, <, \geq, >, =, \neq\}
\end{array}
$$

We now a give an informal description of the meaning of the expressions of the above grammar:

- **constInt** is any integer literal
- **true** and **false** are the unique boolean constants
- **constRef** is a reference to an object in the memory heap
- $\mathcal{E}^{src}$ *op* $\mathcal{E}^{src}$ which stands for an arithmetic expression with any of the arithmetic operators $+, -, div, rem, *$
- $\mathcal{E}^{src}.f$ is a field access expression where the field with name f is accessed
- the cast expression $(Class)\mathcal{E}^{src}$ which is applied only to expressions from a reference type
- the expression **null** stands for the null reference which does not point to any location in the heap
- **this** refers to the current object
- $\mathcal{E}^{src}.m(\mathcal{E}^{src})$ stands for a method invokation expression. Note that here we consider only methods with one argument which return a value
- **new** $Class(\mathcal{E}^{src})$ stands for an object creation expression of class $Class$. We consider only constructors which take only one argument for the sake of readability

The language is also provided with relational expressions, which evaluate to the boolean values:

- $\mathcal{E}^{src}\ \mathcal{R}\ \mathcal{E}^{src}$ where $\mathcal{R} \in \{\leq, <, \geq, >, =, \neq\}$ stands for the relation between two expressions
- $\mathcal{E}^{src}$ **instanceof** $Class$ states that $\mathcal{E}^{src}$ has as type the class $Class$ or one of its subclasses

The expressions can be of object types or basic types. Formally the types are

$$\texttt{JavaType} ::= Class,\ Class \in\ \texttt{ClassTypes} \mid \texttt{int} \mid \texttt{boolean}$$

The next definition gives the control flow constructs of our language as well as the expressions that have a side effect

est-ce que je dois dire qu'on considere un sousensemble de Class qui represente les exceptions ?

Definition 2.2 (Statement) *The grammar for expressions is defined as follows :*

$$
\begin{aligned}
\mathcal{STMT} ::= \ & \mathcal{STMT};\mathcal{STMT} \\
& \mid \texttt{if}\ (\mathcal{E}^{srcRel})\ \texttt{then}\ \{\mathcal{STMT}\}\ \texttt{else}\ \{\mathcal{STMT}\} \\
& \mid \texttt{try}\ \{\mathcal{STMT}\}\ \texttt{catch}\ (Class)\ \{\mathcal{STMT}\} \\
& \mid \texttt{try}\ \{\mathcal{STMT}\}\ \texttt{finally}\ \{\mathcal{STMT}\} \\
& \mid \texttt{try}\ \{\mathcal{STMT}\}\ \texttt{catch}\ (Class)\ \{\mathcal{STMT}\}\ \texttt{finally}\ \{\mathcal{STMT}\} \\
& \mid \texttt{throw}\ \mathcal{E}^{src} \\
& \mid \texttt{while}\ (\mathcal{E}^{srcRel})[\texttt{INV,modif}]\ \ \{\mathcal{STMT}\} \\
& \mid \texttt{return}\ \mathcal{E}^{src} \\
& \mid \texttt{return} \\
& \mid \mathcal{E}^{src} = \mathcal{E}^{src} \\
& \mid \mathcal{E}^{src}
\end{aligned}
$$

From the definition we can see that the language supports also the following constructs :

- $\mathcal{STMT};\mathcal{STMT}$, i.e. statements that execute sequentially
- `if` $(\mathcal{E}^{srcRel})$ `then` $\{\mathcal{STMT}\}$ `else` $\{\mathcal{STMT}\}$ which stands for an if statement. The semantics of the construct is the standard one, i.e. if the relation expression $\mathcal{E}^{srcRel}$ evaluates to true then the statement in the `then` branch is executed, otherwise the statement in the `else` branch is executed
-
-
-
-
-

give the complete explanation

2.1 Source assertion language

The properties that our predicate calculus treats are from first order predicate logic. In the following, we give the formal definition of the assertion language into which the properties are encoded.

Formulas 1 (Definition) *The set of formulas is defined inductively as follows*

$$\begin{array}{ll}
\mathcal{F}^{src} ::= & \psi(\mathcal{E}^{spec}, \mathcal{E}^{spec}) \\
& |T \\
& |\bot \\
& |\mathcal{F}^{src} \wedge \mathcal{F}^{src} \\
& |\mathcal{F}^{src} \vee \mathcal{F}^{src} \\
& |\mathcal{F}^{src} \Rightarrow \mathcal{F}^{src} \\
& |\forall x(\mathcal{F}^{src}(x)) \\
& |\exists x(\mathcal{F}^{src}(x))
\end{array}$$

$$\mathbb{P} ::= \quad ==|\neq|\leq|\leq|\geq|>|<:$$

$$\begin{array}{ll}
\mathcal{E}^{spec} ::= & \mathbf{constInt} \\
& |\ \mathbf{true} \\
& |\ \mathbf{false} \\
& |\ \mathbf{ref} \\
& |\ \mathcal{E}^{spec}\ op\ \mathcal{E}^{spec} \\
& |\ \mathcal{E}^{spec}.f \\
& |\ \mathbf{var} \\
& |\ \mathbf{null} \\
& |\ \mathbf{this} \\
& |\ \backslash \mathtt{typeof}(\mathcal{E}^{spec}) \\
& |\quad \backslash \mathrm{result}
\end{array}$$

Note that the expressions in the assertion language are very similar to the expression in the programming language presented in subsection 2.

We define a function which maps expressions from the programming language into the expressions of the assertion language which is denoted and is typed as follows:

$$\ulcorner . \urcorner^{src2spec} : \mathcal{E}^{src} \rightarrow \mathcal{E}^{spec}$$

The function is defined as follows:

$$\begin{array}{lcl}
\ulcorner \mathbf{constInt} \urcorner^{src2spec} & = & \mathbf{constInt} \\
\ulcorner \mathbf{true} \urcorner^{src2spec} & = & \mathbf{true} \\
\ulcorner \mathbf{false} \urcorner^{src2spec} & = & \mathbf{false} \\
\ulcorner \mathcal{E}^{src} \; op \; \mathcal{E}^{src} \urcorner^{src2spec} & = & \ulcorner \mathcal{E}^{src} \urcorner^{src2spec} \; op \; \ulcorner \mathcal{E}^{src} \urcorner^{src2spec} \\
\ulcorner (Class) \mathcal{E}^{src} \urcorner^{src2spec} & = & \ulcorner \mathcal{E}^{src} \urcorner^{src2spec} \\
\ulcorner \mathcal{E}^{src}.m(\mathcal{E}^{src}) \urcorner^{src2spec} & = & \mathbf{ref} \\
\ulcorner \mathcal{E}^{src}.f \urcorner^{src2spec} & = & \ulcorner \mathcal{E}^{src} \urcorner^{src2spec}.f \\
\ulcorner \mathbf{this} \urcorner^{src2spec} & = & \mathbf{this} \\
\ulcorner \mathbf{new} \; Class(\mathcal{E}^{src}) \urcorner^{src2spec} & = & \mathbf{ref} \\
\ulcorner \mathcal{E}^{src} \; \mathbf{instanceof} \; Class \urcorner^{src2spec} & = & \backslash\texttt{typeof}(\ulcorner \mathcal{E}^{src} \urcorner^{src2spec}) <: Class \wedge \ulcorner \mathcal{E}^{src} \urcorner^{src2spec} \neq \mathbf{null} \\
\ulcorner \mathcal{E}^{src} \; \mathcal{R} \; \mathcal{E}^{src} \urcorner^{src2spec} & = & \ulcorner \mathcal{E}^{src} \urcorner^{src2spec} \mathcal{R} \ulcorner \mathcal{E}^{src} \urcorner^{src2spec}
\end{array}$$

2.2 Weakest Predicate Transformer for the Source Language

The weakest precondition calculates for every statement $\mathcal{STMT}$ from our source language, for any normal postcondition $Post$ and exceptional postcondition function ePost^{src} ($\texttt{Exc} \rightarrow \mathcal{STMT} \rightarrow \mathcal{F}^{src}$), the predicate Pre such that if it holds in the pre state of $\mathcal{STMT}$ and if $\mathcal{STMT}$ terminates normally then $Post$ holds in the poststate and if $\mathcal{STMT}$ terminates on exception Exc then $\mathsf{ePost}^{src}(Exc, \mathcal{STMT})$ holds. The weakest precondition function has the following signature:

$$\mathrm{wp}^{src} : \mathcal{STMT} \rightarrow \mathcal{F}^{src} \rightarrow (\texttt{Exc} \rightarrow \mathcal{STMT} \rightarrow \mathcal{F}^{src}) \rightarrow \mathcal{F}^{src}$$

Before looking at the definition of the weakest predicate transformer we define the exceptional postcondition function ePost^{src}.

2.2.1 Exceptional Postcondition Function

We now look at how the exceptional postconditions for expressions(statements) are managed. As we said the weakest predicate transformer takes into account the normal and exceptional termination of an expression(statement). In both cases the expression(statement) has to satisfy some condition : the normal postcondition in case of normal termination and the exceptional postcondition for exception `Exc` if it terminates on exception `Exc`

We introduce a function ePost^{src} which maps exception types to predicates

$$\mathsf{ePost}^{src} : \; \texttt{ETypes} \; \longrightarrow Predicate$$

The function ePost^{src} returns the predicate $\mathsf{ePost}^{src}(\texttt{Exc})$ that must hold in a particular program point if at this point an exception of type `Exc` is thrown.

We also use function updates for ePost^{src} which are defined in the usual way

$$\mathsf{ePost}^{src}[\oplus\texttt{Exc'} \rightarrow P](\texttt{Exc}, exp) = \begin{cases} P & if \texttt{Exc} <: \texttt{Exc'} \\ \mathsf{ePost}^{src}(\texttt{Exc}, exp) & else \end{cases}$$

2.2.2 Expressions

We define the weakest precondition predicate transformer function over expressions. As we will see in the definition below this definition allows us to get the side effect conditions of the expression evaluationm, namely the conditions for normal and exceptional termination.

- integer and boolean constant access
  ($const \in \{\textbf{constInt}, \textbf{true}, \textbf{false}, \textbf{constRef}\}$)

$$\mathrm{wp}^{src}(\ const\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) = \mathsf{nPost}^{src}$$

- field access expression

$$\begin{array}{l} \mathrm{wp}^{src}(\ \mathcal{E}_1^{src}.f\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) = \\ \mathrm{wp}^{src}(\ \mathcal{E}_1^{src}\ , \\ \qquad \begin{array}{l} \ulcorner \mathcal{E}_1^{src} \urcorner^{src2spec} \neq \textbf{null} \Rightarrow \mathsf{nPost}^{src} \\ \wedge \\ \ulcorner \mathcal{E}_1^{src} \urcorner^{src2spec} = \textbf{null} \Rightarrow \mathsf{ePost}^{src}(\ \texttt{NullPointerExc}, \mathcal{E}_1^{src}) \end{array} , \\ \qquad \mathsf{ePost}^{src}) \end{array}$$

- arithmetic expressions

$$\begin{array}{l} \mathrm{wp}^{src}(\ \mathcal{E}_1^{src}\ op\ \mathcal{E}_2^{src}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) = \\ \mathrm{wp}^{src}(\ \mathcal{E}_1^{src}\ , \mathrm{wp}^{src}(\ \mathcal{E}_2^{src}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}), \mathsf{ePost}^{src}) \end{array}$$

- method invocation

$$
\begin{array}{l}
\mathrm{wp}^{src}(\ \mathcal{E}_1^{src}.m(\mathcal{E}_2^{src})\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) = \\
\mathrm{wp}^{src}(\ \mathcal{E}_1^{src}\ , \mathrm{wp}^{src}(\ \mathcal{E}_2^{src}\ , \\
\qquad \left\{ \begin{array}{l}
\mathcal{E}_1^{src} \neq \ \textbf{null} \Rightarrow \\
\qquad m.\mathsf{Pre}^{src} \begin{array}{l} [\textbf{this} \leftarrow \mathcal{E}_1^{src}] \\ [\mathrm{arg} \leftarrow \mathcal{E}_2^{src}] \end{array} \\
\qquad \wedge \\
\qquad \forall \textbf{ref},\ \forall\ m \in m.\mathsf{modif}^{src} \\
\qquad \left\{ \begin{array}{l}
\backslash\mathtt{typeof}\,(\textbf{ref}) <: m.\mathsf{retType} \wedge \\
m.\mathsf{nPost}^{src} \begin{array}{l} [\ \ \backslash\mathrm{result}\ \ \leftarrow \textbf{ref}] \\ [\textbf{this} \leftarrow \mathcal{E}_1^{src}] \\ [\mathrm{arg} \leftarrow \mathcal{E}_2^{src}] \end{array} \\
\qquad \Rightarrow \mathsf{nPost}^{src}[\ulcorner \mathcal{E}_1^{src}.m(\mathcal{E}_2^{src})\urcorner^{src2spec} \leftarrow \textbf{ref}]
\end{array} \right. \\
\qquad \wedge \\
\qquad \forall \mathtt{E} \in m.\mathsf{exceptions}^{src}, \\
\qquad \forall\ m \in m.\mathsf{modif}^{src} \\
\qquad\ m.\mathsf{exc}^{src}(\mathtt{E}) \Rightarrow \mathsf{ePost}^{src}(\mathtt{E}) \\
\mathcal{E}_1^{src} = \ \textbf{null} \Rightarrow \mathsf{ePost}^{src}(\ \mathtt{NullPntrExc})
\end{array} \right. , \\
\qquad \mathsf{ePost}^{src}), \\
\mathsf{ePost}^{src})
\end{array}
$$

$$
where\ \ulcorner \mathcal{E}_1^{src}.m(\mathcal{E}_2^{src})\urcorner^{src2spec} = \textbf{ref}
$$

- Cast expression

may be give an example

$$
\begin{array}{l}
\mathrm{wp}^{src}(\ (\ \mathtt{Class}\ )\ \mathcal{E}^{src}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) = \\
\mathrm{wp}^{src}(\ \mathcal{E}^{src}\ , \\
\qquad \backslash\mathtt{typeof}\,(\ulcorner \mathcal{E}^{src}\urcorner^{src2spec})\ \mathtt{<:}\ \mathtt{Class}\ \Rightarrow \\
\qquad\qquad \mathrm{wp}^{src}(\ \mathcal{E}^{src}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) \\
\qquad \wedge \qquad\qquad\qquad\qquad\qquad\qquad , \\
\qquad \neg\ \backslash\mathtt{typeof}\,(\ulcorner \mathcal{E}^{src}\urcorner^{src2spec})\ \mathtt{<:}\ \mathtt{Class}\ \Rightarrow \\
\qquad\qquad \mathsf{ePost}^{src}(\ \mathtt{CastExc}, \mathcal{E}^{src}) \\
\qquad \mathsf{ePost}^{src})
\end{array}
$$

- Null expression

$$
\mathrm{wp}^{src}(\ \textbf{null}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) = \mathsf{nPost}^{src}
$$

- this

$$
\mathrm{wp}^{src}(\ \textbf{this}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) = \mathsf{nPost}^{src}
$$

- instance creation

$$\mathrm{wp}^{src}(\ \mathbf{new}\ Class(\mathcal{E}^{src})\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) =$$
$$\mathrm{wp}^{src}(\ \mathcal{E}^{src}\ ,$$

$$\left\{\begin{array}{l}
\mathsf{constr}(Class).\mathsf{Pre}^{src}\ \ [arg \leftarrow \ulcorner \mathcal{E}^{src} \urcorner^{src2spec}] \\
\wedge \\
\forall\ m \in \mathsf{constr}(Class).\mathsf{modif}^{src}, \\
\left.\begin{array}{l}
\ \ \backslash\mathtt{typeof}(\mathbf{ref}) = Class \\
\ \ \wedge \\
\ \ \mathsf{constr}(Class).\mathsf{nPost}^{src}[\mathbf{this} \leftarrow \mathbf{ref}] \\
\ \ [arg \leftarrow \ulcorner \mathcal{E}^{src} \urcorner^{src2spec}]
\end{array}\right\} \Rightarrow \mathsf{nPost}^{src} \\
\wedge \\
\forall \mathtt{Exc} \in \mathsf{constr}(Class).\mathsf{exceptions}^{src}, \\
\forall\ m \in \mathsf{constr}(Class).\mathsf{modif}^{src} \\
\mathsf{constr}(Class).\mathsf{exc}^{src}(\mathtt{Exc}) \Rightarrow \mathsf{ePost}^{src}(\mathtt{Exc})
\end{array}\right. ,$$

$$\mathsf{ePost}^{src})$$

$$where\ \ulcorner \mathbf{new}\ Class(\mathcal{E}^{src}) \urcorner^{src2spec} = \mathbf{ref}$$

Let us see the relational expressions supported in the source programming language

- Instanceof expression

$$\mathrm{wp}^{src}(\ \mathcal{E}^{src}\ instanceof\ Class\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) =$$
$$\mathrm{wp}^{src}(\ \mathcal{E}^{src}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src})$$

- Binary relation over expressions

$$\mathrm{wp}^{src}(\ \mathcal{E}_1^{src}\ \mathcal{R}\ \mathcal{E}_2^{src}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) =$$
$$\mathrm{wp}^{src}(\ \mathcal{E}_1^{src}\ , \mathrm{wp}^{src}(\ \mathcal{E}_2^{src}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}), \mathsf{ePost}^{src})$$

2.2.3 Statements

- integer and boolean constant access

$$\mathrm{wp}^{src}(\ \mathcal{STMT}_1;\mathcal{STMT}_2\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) =$$

$$\mathrm{wp}^{src}(\ \mathcal{STMT}_1\ , \mathrm{wp}^{src}(\ \mathcal{STMT}_2\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}), \mathsf{ePost}^{src})$$

- assignment
  - local variable assignemnt

$$\mathrm{wp}^{src}(\ \mathcal{E}_1^{src} = \mathcal{E}_2^{src}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) =$$

$$\begin{array}{l}\mathrm{wp}^{src}(\ \mathcal{E}_2^{src}\ , \\
\quad \mathrm{wp}^{src}(\ \mathcal{E}_1^{src}\ , \mathsf{nPost}^{src}[\ulcorner \mathcal{E}_1^{src} \urcorner^{src2spec} \leftarrow \ulcorner \mathcal{E}_2^{src} \urcorner^{src2spec}], \mathsf{ePost}^{src}), \\
\quad \mathsf{ePost}^{src})\end{array}$$

– instance field assignemnt

$$\mathrm{wp}^{src}(\ \mathcal{E}_1^{src}.f = \mathcal{E}_2^{src}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) =$$

$$\begin{array}{l}
\mathrm{wp}^{src}(\ \mathcal{E}_1^{src}\ , \\
\quad \mathrm{wp}^{src}(\ \mathcal{E}_2^{src}\ ,\ \begin{array}{l} \mathbf{null} \neq \ulcorner \mathcal{E}_1^{src} \urcorner^{src2spec} \Rightarrow \\ \qquad \mathsf{nPost}^{src}[f \leftarrow f \oplus [\ulcorner \mathcal{E}_1^{src} \urcorner^{src2spec} \rightarrow \ulcorner \mathcal{E}_2^{src} \urcorner^{src2spec}]] \\ \wedge \\ \mathbf{null} = \ulcorner \mathcal{E}_1^{src} \urcorner^{src2spec} \Rightarrow \\ \qquad \mathsf{ePost}^{src}(\mathtt{NullPointerExc}) \end{array} , \\
\quad \mathsf{ePost}^{src}), \\
\quad \mathsf{ePost}^{src})
\end{array}$$

- if statement

$$\mathrm{wp}^{src}(\ \begin{array}{l} \mathtt{if}\ (\mathcal{E}^{src}) \\ \mathtt{then}\{\mathcal{STMT}_1\} \\ \mathtt{else}\ \{\mathcal{STMT}_2\} \end{array}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) =$$

$$\begin{array}{l}
\mathrm{wp}^{src}(\ \mathcal{E}^{srcRel}\ , \\
\quad \begin{array}{l} \ulcorner \mathcal{E}^{srcRel} \urcorner^{src2spec} \Rightarrow \mathrm{wp}^{src}(\ \mathcal{STMT}_1\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) \\ \wedge \\ \neg\ \ulcorner \mathcal{E}^{srcRel} \urcorner^{src2spec} \Rightarrow \mathrm{wp}^{src}(\ \mathcal{STMT}_2\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) \end{array} , \\
\quad \mathsf{ePost}^{src})
\end{array}$$

- throw exceptions

$$\mathrm{wp}^{src}(\ \mathtt{throw}\ \mathcal{E}^{src}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) =$$

$$\begin{array}{l}
\mathrm{wp}^{src}(\ \mathcal{E}^{src}\ , \\
\quad \begin{array}{l} \ulcorner \mathcal{E}^{src} \urcorner^{src2spec} \neq \mathbf{null} \Rightarrow \mathsf{ePost}^{src}(\backslash\mathtt{typeof}(\mathcal{E}^{src})) \\ \ulcorner \mathcal{E}^{src} \urcorner^{src2spec} = \mathbf{null} \Rightarrow \mathsf{ePost}^{src}(\mathtt{NullPointerExc}) \end{array} , \\
\quad \mathsf{ePost}^{src})
\end{array}$$

- try catch statement

$$\mathrm{wp}^{src}(\ \mathtt{try}\ \{\mathcal{STMT}_1\}\ \mathtt{catch}(\mathtt{Exc}\ c)\ \{\mathcal{STMT}_2\}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) =$$

$$\begin{array}{l}
\mathrm{wp}^{src}(\ \mathcal{STMT}_1\ , \\
\quad \mathsf{nPost}^{src}, \\
\quad \mathsf{ePost}^{src} \oplus [\mathtt{Exc} \longrightarrow \mathrm{wp}^{src}(\ \mathcal{STMT}_2\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src})])
\end{array}$$

- try finally

$$\mathrm{wp}^{src}(\ \mathtt{try}\ \{\mathcal{STMT}_1\}\ \mathtt{finally}\ \{\mathcal{STMT}_2\}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) =$$

$$\begin{array}{l}
\mathrm{wp}^{src}(\ \mathcal{STMT}_1\ , \\
\quad \mathrm{wp}^{src}(\ \mathcal{STMT}_2\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}), \\
\quad \mathsf{ePost}^{src} \oplus [\mathtt{Exception} \longrightarrow \mathrm{wp}^{src}(\ \mathcal{STMT}_2\ , \mathsf{ePost}^{src}(\mathtt{Exception}), \mathsf{ePost}^{src})])
\end{array}$$

where *exc* is the exception object thrown by $\mathcal{STMT}_1$.

- try catch finally

$$\mathrm{wp}^{src}\left(\begin{array}{l} \texttt{try}\ \{\mathcal{STMT}_1\} \\ \texttt{catch}(Class\ c)\ \{\mathcal{STMT}_2\} \\ \texttt{finally}\ \{\mathcal{STMT}_3\} \end{array}, \mathsf{nPost}^{src}, \mathsf{ePost}^{src}\right)$$
$$=$$
$$\mathrm{wp}^{src}\left(\begin{array}{l} \texttt{try}\ \{\texttt{try}\ \{\mathcal{STMT}_1\}\texttt{catch}(Class\ c)\ \{\mathcal{STMT}_2\}\} \\ \texttt{finally}\ \{\mathcal{STMT}_3\} \end{array}, \mathsf{nPost}^{src}, \mathsf{ePost}^{src}\right)$$

- loop statement

$$\mathrm{wp}^{src}(\ \texttt{while}\ (\mathcal{E}^{src})\ [\texttt{INV}, \texttt{modif}]\ \ \{\mathcal{STMT}\}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) =$$

$$\begin{array}{l} \texttt{INV} \wedge \\ \forall\ m, m \in \texttt{modif}, \\ \quad \texttt{INV} \Rightarrow \\ \quad\quad \mathrm{wp}^{src}(\ \mathcal{E}^{src}\ , \\ \quad\quad\quad \begin{array}{l} \ulcorner \mathcal{E}^{src} \urcorner^{src2spec} = \mathbf{true} \Rightarrow\ \mathrm{wp}^{src}(\ \mathcal{STMT}\ , \texttt{INV}, \mathsf{ePost}^{src}) \\ \ulcorner \mathcal{E}^{src} \urcorner^{src2spec} = \mathbf{false} \Rightarrow \mathsf{nPost}^{src} \end{array}, \\ \quad\quad \mathsf{ePost}^{src}) \end{array}$$

where

- $\texttt{INV}$ is the invariant of the loop
- $m_i.i = 1..k$ are the locations that may be modified by a loop

- return statement

$$\begin{array}{l} \mathrm{wp}^{src}(\ \texttt{return}\ \mathcal{E}^{src}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) = \\ \mathrm{wp}^{src}(\ \mathcal{E}^{src}\ , \mathsf{nPost}^{src}[\quad \backslash\mathrm{result}\ \ \leftarrow \ulcorner \mathcal{E}^{src} \urcorner^{src2spec}], \mathsf{ePost}^{src}) \end{array}$$

where \result is a specification variable that can be met in the postcondition and denotes to the value returned of a non void method

3 Bytecode

3.1 Introduction

In the following, we consider the bytecode language and its semantics introduced in Chapter **??**, Section **??**. However, in this section we will give a different axiomatics semantics which this time will take advantage of the compiler definition.

3.2 Weakest predicate transformer for Bytecode language

4 Compiler

We now turn to specify a simple compiler from the source language presented in Section 2 into the bytecode language. The compiler does not perform any optimizations.

the exception handler function

4.1 Compiling source formulas

In the previous section, we have seen how source statements are compiled into a sequence of bytecode instructions. We now look at how formulas referring to source expressions are compiled into formulas that "talk" about bytecode expressions. These formulae appear in the specification of a source component. The compiler function is $\ulcorner . \urcorner$ and has the signature :

$$\ulcorner . \urcorner : \mathcal{F}^{src} \rightarrow \mathcal{F}^{bc}$$

Definition 1 (Formula compiler)

$$\ulcorner \mathcal{F}^{src} \urcorner = \begin{array}{ll} \psi(\ulcorner \mathcal{E}_1^{src} \urcorner^{spec}, \ulcorner \mathcal{E}_2^{src} \urcorner^{spec}) & \psi \in \mathbb{P},\ \mathcal{E}_1^{src}, \mathcal{E}_2^{src} \in \mathcal{E}^{src} \\ T & if\ \mathcal{F}^{src} = T \\ \perp & if\ \mathcal{F}^{src} = \perp \\ \ulcorner \mathcal{F}_1^{src} \urcorner \wedge \ulcorner \mathcal{F}_2^{src} \urcorner & if\ \mathcal{F}^{src} = \mathcal{F}_1^{src} \wedge \mathcal{F}_2^{src} \\ \ulcorner \mathcal{F}_1^{src} \urcorner \vee \ulcorner \mathcal{F}_2^{src} \urcorner & if\ \mathcal{F}^{src} = \mathcal{F}_1^{src} \vee \mathcal{F}_2^{src} \\ \ulcorner \mathcal{F}_1^{src} \urcorner \Rightarrow \ulcorner \mathcal{F}_2^{src} \urcorner & if\ \mathcal{F}^{src} = \mathcal{F}_1^{src} \Rightarrow \mathcal{F}_2^{src} \\ \forall\ x\ . \ulcorner \mathcal{F}_1^{src} \urcorner & if\ \mathcal{F}^{src} = \forall\ x.(\mathcal{F}_1^{src}) \\ \exists\ x\ . \ulcorner \mathcal{F}_1^{src} \urcorner & if\ \mathcal{F}^{src} = \exists\ x.(\mathcal{F}_1^{src}) \end{array}$$

Note that in the compilation of atomic predicates we compile the expressions with $\ulcorner . \urcorner^{spec}$ which compiles the identifiers in the expressions to the corresponding identifier in the bytecode. The function $\ulcorner . \urcorner^{spec}$ is described in [1]. We illustrate the effect of $\ulcorner . \urcorner^{spec}$ with an example:

```
public class B{
  //@ requires a.b != null
  public int m (A a) {
    ...
  }
}
```

The application of $\ulcorner . \urcorner^{spec}$ to the precondition is of the form

$$\ulcorner a.b! = null \urcorner^{spec} = \texttt{reg}_1.\texttt{cpIndex}(\texttt{b})! = \texttt{null}$$

where $\texttt{reg}_1$ is a register of method `m` in which the parameter `a` is stored and `cpIndex(b)` is the index of the field `b` of class `A` in the constant pool of class `B`.

An easy to see property is the following property (the proof can be done inductively over the formula structure) :

Property 1 (Compiler Property 1)

$$\mathcal{F}^{src} =^{mod\ Names\ and\ bools} \ulcorner \mathcal{F}^{src} \urcorner$$

Also, as the source language does not contain stack expressions (`st(cntr )` and `cntr`) and because of the definition of $\mathcal{F}^{src}$ no formula $\psi \in \mathcal{F}^{src}$ contains stack expressions. From the compiler function, we can then obtain the second property about the compiler :

Property 2 (Compiler Property 2) *$\forall \psi \in \mathcal{F}^{src}$. $\ulcorner \psi \urcorner$ does not contain stack expressions*

Another evdient point is that the set of formulas on bytecode level $\mathcal{F}^{bc}$ is larger than the set of source formulas and thus not all bytecode formulas have their corresponding image in $\mathcal{F}^{src}$. This is due to the fact that in $\mathcal{F}^{bc}$ there are formulas that mention stack expressions but those expressions do not have a counterpart on source level. Thus, we can characterise the domain of $\ulcorner . \urcorner$ with the following property:

Property 3 (Compiler Property 3) *$\mathcal{F}^{bc}_{no\ stack}$ is the subset of formulas $\psi^{bc} \in \mathcal{F}^{bc}$ that do not contain stack expressions. $\forall \psi^{bc} \in \mathcal{F}^{bc}_{no\ stack}\ \exists \psi^{src} \in \mathcal{F}^{src}. \ulcorner \psi^{src} \urcorner = \psi^{bc}$*

4.2 Compiling expressions in bytecode instructions

We now turn to the definition of the compiler from the source language defined in Section 2. The compiler function is denoted with $\ulcorner\urcorner$ and its signature is :

$$\ulcorner\urcorner : \mathcal{STMT} \longrightarrow bytecode$$

Although expressions on source level can be atomic, this is not the case for their bytecode compilation, i.e. an expression can be compiled in several instructions.

- integer or boolean constant access
  - integer constant access

$$\ulcorner \mathbf{constInt} \urcorner = \ \texttt{push} \quad \mathbf{constInt}$$

  - boolean constant access

$$\ulcorner \mathbf{true} \urcorner = \ \texttt{push} \quad 1$$

$$\ulcorner \mathbf{false} \urcorner = \ \texttt{push} \quad 0$$

  Note: the source boolean expressions are compiled down to integers

- method invokation

$$\ulcorner \mathcal{E}_1^{src}.m(\mathcal{E}_2^{src}) \urcorner = \begin{array}{l} \ulcorner \mathcal{E}_1^{src} \urcorner; \\ \ulcorner \mathcal{E}_2^{src} \urcorner; \\ \texttt{invoke} \ \ m \end{array}$$

- field access

$$\ulcorner \mathcal{E}^{src}.f \urcorner = \begin{array}{l} \ulcorner \mathcal{E}^{src} \urcorner; \\ \texttt{getfield} \ \ f \end{array}$$

- local variable access

$$\ulcorner \mathbf{var} \urcorner = \ \texttt{load} \ \ \texttt{reg}_\texttt{i}$$

  where $\texttt{reg}_\texttt{i}$ is the local variable at index i

- arithmetic expressions

$$\ulcorner \mathcal{E}_1^{src}\ op\ \mathcal{E}_2^{src} \urcorner = \begin{array}{l} \ulcorner \mathcal{E}_1^{src} \urcorner; \\ \ulcorner \mathcal{E}_2^{src} \urcorner; \\ \texttt{op} \end{array}$$

- cast expression

$$\ulcorner (\texttt{ Class})\ \mathcal{E}^{src} \urcorner = \begin{array}{l} \ulcorner \mathcal{E}^{src} \urcorner; \\ \texttt{checkCast} \quad \texttt{Class}\ ; \end{array}$$

Note : for Java Sun compiler, this compilation is done if this is a down cast (in case this is an up cast no checkcast is generated). where the execution of the compilation of $\mathcal{E}^{src}$ affects the stack :

- instanceof expression

$$\ulcorner \mathcal{E}^{src}\ \textbf{instanceof}\ Class \urcorner = \begin{array}{l} \ulcorner \mathcal{E}^{src} \urcorner; \\ \texttt{instanceof} \quad Class; \end{array}$$

- null expression

$$\ulcorner \textbf{null} \urcorner = \texttt{push} \quad \textbf{null}$$

- object creation

$$\ulcorner \textbf{new}\ Class(\mathcal{E}^{src}) \urcorner = \begin{array}{l} \texttt{new} \quad Class; \\ \texttt{dup}\ ; \\ \ulcorner \mathcal{E}^{src} \urcorner; \\ \texttt{invoke} \quad \mathsf{constr}(Class); \end{array}$$

- this instance

$$\ulcorner \textbf{this} \urcorner = \texttt{load} \quad \texttt{reg}_0$$

4.3 Compiling control statements in bytecode instructions

- compositional statement

$$\ulcorner \mathcal{STMT}_1; \mathcal{STMT}_2 \urcorner = \begin{array}{l} \ulcorner \mathcal{STMT}_1 \urcorner; \\ \ulcorner \mathcal{STMT}_2 \urcorner \end{array}$$

- if statement

$$\ulcorner \texttt{if}\ (\mathcal{E}^{src})\ \texttt{then}\ \{\mathcal{STMT}_1\}\ \texttt{else}\ \{\mathcal{STMT}_2\} \urcorner = \begin{array}{l} \ulcorner \mathcal{E}^{src} \urcorner; \\ \texttt{if\_cond} \quad l_{true}; \\ \ulcorner \mathcal{STMT}_2) \urcorner \\ \texttt{goto} \quad l; \\ l_{true} : \ulcorner \mathcal{STMT}_1) \urcorner \\ l : \end{array}$$

- assignment statement. We consider two cases - assignement to instance fields and assignemnts to method local variables and parameters.

– field assignement. The expressions of the form f = v, where f is an instance field of this object are desugared to this.f = v.

$$\ulcorner \mathcal{E}_1^{src}.f = \mathcal{E}_2^{src} \urcorner = \begin{array}{l} \ulcorner \mathcal{E}_1^{src} \urcorner; \\ \ulcorner \mathcal{E}_2^{src} \urcorner; \\ \texttt{putfield}\ \ f; \end{array}$$

– method local variable or parameter update

$$\ulcorner \mathbf{var} = \mathcal{E}^{src} \urcorner = \begin{array}{l} \ulcorner \mathcal{E}^{src} \urcorner; \\ \texttt{store}\ \ \texttt{reg}_\texttt{i}; \end{array}$$

- try catch statement

$$\ulcorner \texttt{try}\ \{\mathcal{STMT}_1\}\ \texttt{catch}\ (Class\ name)\{\mathcal{STMT}_2\}\urcorner =$$

$\ulcorner \mathcal{STMT}_1 \urcorner$;
goto l;
$\ulcorner \mathcal{STMT}_2 \urcorner$;
goto l;
...
l :

addExcHandler(startInd($\ulcorner \mathcal{STMT}_1 \urcorner$), endInd($\ulcorner \mathcal{STMT}_1 \urcorner$), startInd($\ulcorner \mathcal{STMT}_2 \urcorner$), $Class$))

The compiler compiles the normal statement $\mathcal{STMT}_1$ and the exception handler $\mathcal{STMT}_2$. Then in the exception handler table a new element is added - it describes that the handler starting at startInd($\ulcorner \mathcal{STMT}_2 \urcorner$) protects the region from startInd($\ulcorner \mathcal{STMT}_1 \urcorner$) to endInd($\ulcorner \mathcal{STMT}_1 \urcorner$) from exceptions of type $Class$.

- try finally statement

$$\ulcorner \texttt{try}\ \{\mathcal{STMT}_1\}\ \texttt{finally}\ \{\mathcal{STMT}_2\}\urcorner =$$

$\ulcorner \mathcal{STMT}_1 \urcorner$;
`jsr` s;
`goto` l;

{ default exception handler}
h : `astore` e;
`jsr` s;
`aload` e;
`athrow` ;

{ compilation of the subroutine}
s : `astore` k;
$\ulcorner \mathcal{STMT}_2 \urcorner$;
`ret` k

l : ...

addExcHandler(startInd($\ulcorner \mathcal{STMT}_1 \urcorner$), endInd($\ulcorner \mathcal{STMT}_1 \urcorner$), h, $Exception$))

We keep close to the JVM (short for Java Virtual Machine) specification, which requires that the subroutines must be compiled using `jsr` and `ret` instructions. The `jsr` actually jumps to the first instruction of the compiled subroutine which starts at index s and pushes on the operand stack the index of the next instruction of the `jsr` that caused the execution of the subroutine. The first instruction of the compilation of the subroutine stores the stack top element in the local variable at index k (i.e. stores in the local variable at index k the index of the instruction following the `jsr` instruction). Thus, after the code of the subroutine is executed, the `ret k` instruction jumps to the instruction following the corresponding `jsr` .

Note:

1. we assume that the local variable e and k are not used in the compilation of the statement $\mathcal{STMT}_1$.
2. here we also assume that the statement $\mathcal{STMT}_1$ does not contain a `return` instruction

The compiler adds a default exception handler whose implementation guarantees that in exceptional termination case, the subroutine is also executed. The exception handler is added in the exception handler table. It protects the instructions of the statement $\ulcorner \mathcal{STMT}_1 \urcorner$ against any thrown exception of type or subtype *Exception*.

- try catch finally statement

$$\ulcorner \texttt{try}\ \{\mathcal{STMT}_1\}\ \texttt{catch}\ (Class)\ \{\mathcal{STMT}_2\}\ \texttt{finally}\ \{\mathcal{STMT}_3\} \urcorner =$$

$$\ulcorner \texttt{try}\ \{\texttt{try}\ \{\mathcal{STMT}_1\}\ \texttt{catch}\ (Class)\ \{\mathcal{STMT}_2\}\ \}\ \texttt{finally}\ \{\mathcal{STMT}_3\} \urcorner$$

- throw exception statement

$$\ulcorner \texttt{throw}\ \mathcal{E}^{src} \urcorner = \begin{array}{l} \ulcorner \mathcal{E}^{src} \urcorner; \\ \texttt{athrow}\ ; \end{array}$$

- loop statement

$$\begin{array}{l} \ulcorner \texttt{while}\ (\mathcal{E}^{srcRel})[\texttt{INV}, \texttt{modif}]\ \ \{\mathcal{STMT}\} \urcorner = \\ \quad \texttt{goto}\ \ loopEntry; \\ \ loopBody : \ulcorner \mathcal{STMT} \urcorner; \\ \ [\ulcorner \texttt{INV} \urcorner^{spec}, \ulcorner \texttt{modif} \urcorner^{spec}] \\ \ loopEntry : \ulcorner \mathcal{E}^{src} \urcorner; \\ \quad \texttt{if\_cond}\ \ loopBody; \end{array}$$

- return statement

$$\ulcorner \texttt{return}\ \mathcal{E}^{src} \urcorner = \begin{array}{l} \ulcorner \mathcal{E}^{src} \urcorner; \\ \texttt{return} \end{array}$$

4.4 Properties of the compiler function

A property that can be established for the compiler is the following:

Property 1 *For any statement $\mathcal{STMT}$, the compilation $\ulcorner \mathcal{STMT} \urcorner$ does not contain jump instructions (* `goto` *,* `if_cond` *) outside $\ulcorner \mathcal{STMT} \urcorner$ except possibly for the last instruction in $\ulcorner \mathcal{STMT} \urcorner$.*

The property can be established by structural induction of the compilation $\ulcorner \mathcal{STMT} \urcorner$

5 Compiling Proof Obligations

We turn now to study the relationship between the proof obligations on source and bytecode level. We show that syntactically the proof obligations are the same modulo names and some types.

5.1 Auxiliary Properties

Before stating the main theorem we need some auxiliary properties. First, we establish that adding a `goto` instruction to a sequence of instructions does not change the weakest predicate of the augmented bytecode sequence.

Lemma 1 *Let's have the sequence of bytecode instructions $i_1; ...; i_k$ where $next(i_k) = i_l$*

$$wp^{bc}(\ i_1; ...; i_k\ , \psi, \psi^{bc}_{exc}) = wp^{bc}(\ i_1; ...; i_k;\ \texttt{goto}\ l\ , \psi, \psi^{bc}_{exc})$$

The proof is based on the fact that the instruction `goto` *does not have side effects and thus, the following holds: $wp^{bc}(\ \texttt{goto l}\ , \psi, \psi^{bc}_{exc}) = \psi$*

We now turn to see how the execution of the compilation $\ulcorner \mathcal{E}^{src} \urcorner$ of an expression $\mathcal{E}^{src}$ affects the operand stack. In particular, we claim that if the execution of the compiled expression $\ulcorner \mathcal{E}^{src} \urcorner$ terminates normally then the stack top contains the value of the expression $\ulcorner \mathcal{E}^{src} \urcorner^{spec}$. This actually reflects how we expect that the virtual machine execute bytecode programs.

This fact in terms of weakest preconditons can be expressed as follows:

Lemma 2 (Wp of a compiled expression) *For any expression $\mathcal{E}^{src}$ from our source language, for any formula $\psi : \mathcal{F}^{src}$ of the source assertion language and any formula $\phi : \mathcal{F}^{bc}$ such that ϕ may only contain stack expressions of the form* `st(cntr - k)` *, $k \geq 0$, there exist $Q, R : \mathcal{F}^{src}$ such that the following holds*

- $$\begin{array}{l} wp^{src}(\ \mathcal{E}^{src}\ , \psi, \psi^{bc}_{exc})\ \equiv \\ Q \Rightarrow \psi \\ \wedge \\ R \end{array}$$

- $$\begin{array}{l} wp^{bc}(\ \ulcorner \mathcal{E}^{src} \urcorner\ , \psi, \ulcorner \psi^{bc}_{exc} \urcorner)\ \equiv \\ \ulcorner Q \urcorner^{spec} \Rightarrow \phi \begin{array}{l}[\texttt{cntr} \leftarrow \texttt{cntr} + 1] \\ [\texttt{st(cntr +1)} \leftarrow \ulcorner \mathcal{E}^{src} \urcorner^{spec}]\end{array} \\ \\ \wedge \\ \ulcorner R \urcorner^{spec} \end{array}$$

We proceed with several cases of the proof, which is done by induction over the structure of the formula

Proof :

1. $\mathcal{E}^{src} = const, const \in$ **constInt**, **true**, **false**

$\{$ *source case* $\}$
(1)$\mathrm{wp}^{src}($ *const* $, \psi, \psi^{bc}_{exc})$
$\{$ *following the definition of the wp function for source expressions in subsection 2.2* $\}$
$\equiv \psi$

$\{$ *bytecode case* $\}$
(2)$\mathrm{wp}^{bc}(\ulcorner const \urcorner , \phi, \ulcorner \psi^{bc}_{exc} \urcorner)$
$\{$ *following the definition of the compiler function in subsection 4.2* $\}$
$\equiv \mathrm{wp}^{bc}($ `push` $\ulcorner const \urcorner^{spec} , \phi, \ulcorner \psi^{bc}_{exc} \urcorner)$
$\{$ *following the definition of the wp function for bytecode in subsection 3.2* $\}$
$$\equiv \phi \begin{array}{l} [\texttt{cntr} \leftarrow \texttt{cntr} + 1] \\ [\texttt{st(cntr +1)} \leftarrow \ulcorner const \urcorner^{spec}] \end{array}$$

$\{$ *from (1) and (2) and* $Q, R = T$ *this case holds* $\}$

2. $\mathcal{E}^{src} = \mathcal{E}^{src}.f$

$\{$ *source case* $\}$
(1)$\mathrm{wp}^{src}(\ \mathcal{E}^{src}.f\ , \psi, \psi^{bc}_{exc})$
$\{$*following the definition of the wp function for source expressions in subsection 2.2* $\}$

$$\equiv \mathrm{wp}^{src}\left(\ \mathcal{E}^{src}\ ,\ \begin{array}{l} \ulcorner \mathcal{E}^{src} \urcorner^{src2spec} \neq \mathbf{null} \Rightarrow \psi \\ \wedge \\ \ulcorner \mathcal{E}^{src} \urcorner^{src2spec} \neq \mathbf{null} \Rightarrow \psi^{bc}_{exc}(\ \texttt{NullPntrExc}) \end{array}, \psi^{bc}_{exc}\right)$$

$\{$ *bytecode case* $\}$
(2)$\mathrm{wp}^{bc}(\ \ulcorner \mathcal{E}^{src}.f \urcorner\ , \phi, \ulcorner \psi^{bc}_{exc} \urcorner)$
$\{$ *following the definition of the compiler function in subsection 4.2* $\}$

$$\equiv \mathrm{wp}^{bc}\left(\ \begin{array}{l} \ulcorner \mathcal{E}^{src} \urcorner; \\ \texttt{getfield}\ \ f \end{array}\ , \phi, \psi^{bc}_{exc}\right)$$

$\{$*following the definition of the wp function for bytecode in subsection 3.2* $\}$

$$\equiv \mathrm{wp}^{bc}\left(\ \ulcorner \mathcal{E}^{src} \urcorner\ ,\ \begin{array}{l} \texttt{st(cntr)} \neq \mathbf{null} \Rightarrow \\ \phi[\texttt{st(cntr)} \leftarrow f(\texttt{st(cntr)})] \\ \wedge \\ \texttt{st(cntr)} = \mathbf{null} \Rightarrow \ulcorner \psi^{bc}_{exc} \urcorner(\ \texttt{NullPntrExc}) \end{array}, \ulcorner \psi^{bc}_{exc} \urcorner\right)$$

$\{$ *from (1) and (2) we apply the induction hypothesis* $\}$
$\exists Q', R' : \mathcal{F}^{src},$

$$(3)\ \mathrm{wp}^{src}\left(\ \mathcal{E}^{src}\ ,\ \begin{array}{l} \ulcorner \mathcal{E}^{src} \urcorner^{src2spec} \neq \mathbf{null} \Rightarrow \psi \\ \wedge \\ \ulcorner \mathcal{E}^{src} \urcorner^{src2spec} \neq \mathbf{null} \Rightarrow \psi^{bc}_{exc}(\ \texttt{NullPntrExc}) \end{array}, \psi^{bc}_{exc}\right)$$

$\equiv$

$$Q' \Rightarrow \begin{array}{l} \ulcorner \mathcal{E}^{src} \urcorner^{src2spec} \neq \mathbf{null} \Rightarrow \psi \\ \wedge \\ \ulcorner \mathcal{E}^{src} \urcorner^{src2spec} \neq \mathbf{null} \Rightarrow \psi^{bc}_{exc}(\ \texttt{NullPntrExc}) \end{array}$$

$\wedge$
R'

$$(4)\ \mathrm{wp}^{bc}\left(\ \ulcorner \mathcal{E}^{src} \urcorner\ ,\ \begin{array}{l} \texttt{st(cntr)} \neq \mathbf{null} \Rightarrow \\ \phi[\texttt{st(cntr)} \leftarrow f(\texttt{st(cntr)})] \\ \wedge \\ \texttt{st(cntr)} = \mathbf{null} \Rightarrow \ulcorner \psi^{bc}_{exc} \urcorner(\ \texttt{NullPntrExc}) \end{array}, \ulcorner \psi^{bc}_{exc} \urcorner\right)$$

$\equiv$

$$\ulcorner Q' \urcorner \Rightarrow \begin{array}{l} \texttt{st(cntr)} \neq \mathbf{null} \Rightarrow \phi \\ \wedge \\ \texttt{st(cntr)} \neq \mathbf{null} \Rightarrow \ulcorner \psi^{bc}_{exc} \urcorner(\ \texttt{NullPntrExc}) \end{array} \begin{array}{l} [\texttt{cntr} \leftarrow \texttt{cntr} + 1] \\ [\texttt{st(cntr + 1)} \leftarrow \ulcorner \mathcal{E}^{src} \urcorner^{spec}] \end{array}$$

$\wedge$
$\ulcorner R' \urcorner$
$\equiv$

$$\ulcorner Q' \urcorner \Rightarrow \begin{array}{l} \ulcorner \mathcal{E}^{src} \urcorner^{spec} \neq \mathbf{null} \Rightarrow \phi \begin{array}{l} [\mathtt{cntr} \ \leftarrow \mathtt{cntr} \ +1] \\ [\mathtt{st(cntr + 1)} \ \leftarrow \ulcorner \mathcal{E}^{src} \urcorner^{spec}] \end{array} \\ \wedge \\ \ulcorner \mathcal{E}^{src} \urcorner^{spec} \neq \mathbf{null} \Rightarrow \ulcorner \psi^{bc}_{exc} \urcorner (\ \mathtt{NullPntrExc}) \end{array}$$

$$\wedge$$

$$\ulcorner R' \urcorner$$

{ *from (3) and (4) this case holds* }

5.2 Proof obligation equivalence

Theorem 1 *For every statement $\mathcal{STMT}$ from the source language, any formula $\psi \in \mathcal{F}^{src}$ and any exceptional postcondition function ePost^{src} and ψ^{bc}_{exc} such that $\mathsf{ePost}^{src}(\ \mathtt{Exc}, \mathcal{E}^{src}) =^{mod \ Names \ and \ bools} \psi^{bc}_{exc}(\ \mathtt{Exc}, \ulcorner \mathcal{E}^{src} \urcorner)$ we have that*

- $$wp^{bc}(\ \ulcorner \mathcal{STMT} \urcorner \ , \ulcorner \psi \urcorner, \psi^{bc}_{exc})$$

  does not contain subexpressions of the form `st(ind)` *and* `cntr`

- $$wp^{src}(\ \mathcal{STMT} \ , \psi, \mathsf{ePost}^{src}) =^{mod \ Names \ and \ bools} wp^{bc}(\ \ulcorner \mathcal{STMT} \urcorner \ , \ulcorner \psi \urcorner, \psi^{bc}_{exc})$$

Proof:
By structural induction over the structure of the source expressions and statements

Note: in the following, we are using the following property of the wp predicate transformer, namely

$$\mathrm{wp}(\mathcal{STMT}, \psi, \psi^{\mathrm{bc}}_{\mathrm{exc}}) \wedge \mathrm{wp}(\mathcal{STMT}, \phi, \psi^{\mathrm{bc}}_{\mathrm{exc}}) = \mathrm{wp}(\mathcal{STMT}, \psi \wedge \phi, \psi^{\mathrm{bc}}_{\mathrm{exc}})$$

which is easy to establish.

Expressions

integer constant access

{ *by definition* }

$$wp^{src}(\ const \ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) = \mathsf{nPost}^{src}$$

{ *from the definition of the compiler function and the weakest precondition over bytecode* }

$$wp^{bc}(\ \ulcorner const \urcorner \ , \ulcorner \mathsf{nPost}^{src} \urcorner, \psi^{bc}_{exc})$$

$$=$$

$$wp^{bc}(\ \ \mathtt{push} \ \ \ \ulcorner const \urcorner^{spec} \ , \ulcorner \mathsf{nPost}^{src} \urcorner, \psi^{bc}_{exc})$$

{ *from the definition of the weakest precondition function of* `push` }

$$wp^{bc}(\ \texttt{push}\ \ \ulcorner \text{eval}(const)\urcorner\ , \ulcorner \mathsf{nPost}^{src}\urcorner, \psi^{bc}_{exc})$$
$$=$$
$$\ulcorner \mathsf{nPost}^{src}\urcorner[\texttt{cntr}\ \leftarrow \texttt{cntr}\ + 1][\texttt{st(cntr + 1)}\ \leftarrow const]$$

{ $\ulcorner \mathsf{nPost}^{src}\urcorner$ *does not contain stack and stack counter expressions from Property 2 on page 11* }

$$\ulcorner \mathsf{nPost}^{src}\urcorner[\texttt{cntr}\ \leftarrow \texttt{cntr}\ + 1][\texttt{st(cntr + 1)}\ \leftarrow const]$$
$$=$$
$$\ulcorner \mathsf{nPost}^{src}\urcorner$$

{ *from the compiler for formulas we know that* $\forall \psi\ .\ \psi =^{mod\ Names\ and\ bools} \ulcorner \psi \urcorner$ }

$$\mathsf{nPost}^{src} =^{mod\ Names\ and\ bools} \ulcorner \mathsf{nPost}^{src}\urcorner$$

method invocation

assignment expressions

local variable assignment

{ *by definition of the weakest precondition for assignment* }

(0)

$$wp^{src}(\ \mathcal{E}^{src}_1 = \mathcal{E}^{src}_2\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) =$$

$$wp^{src}(\ \mathcal{E}^{src}_2\ , \mathsf{nPost}^{src}[\mathcal{E}^{src}_1 \leftarrow \ulcorner \mathcal{E}^{src}_2 \urcorner^{spec}], \mathsf{ePost}^{src})$$

{ *by defintion of the compiler* }

(1)

$$wp^{bc}(\ \ulcorner \mathcal{E}^{src}_1 = \mathcal{E}^{src}_2 \urcorner\ , \ulcorner \mathsf{nPost}^{src}\urcorner, \psi^{bc}_{exc})$$
$$=$$
$$wp^{bc}(\ \begin{array}{l} \ulcorner \mathcal{E}^{src}_2 \urcorner; \\ \texttt{store}\ \ \ulcorner \mathcal{E}^{src}_1 \urcorner; \end{array}\ , \ulcorner \mathsf{nPost}^{src}\urcorner, \psi^{bc}_{exc})$$

{ *by defintion of the weakest precondition function for* `store` }

$$wp^{bc}(\ \begin{array}{l} \ulcorner \mathcal{E}^{src}_2 \urcorner; \\ \texttt{store}\ \ \ulcorner \mathcal{E}^{src}_1 \urcorner; \end{array}\ , \ulcorner \mathsf{nPost}^{src}\urcorner, \psi^{bc}_{exc})$$
$$=$$
$$wp^{bc}(\ \ulcorner \mathcal{E}^{src}_2 \urcorner\ , \ulcorner \mathsf{nPost}^{src}\urcorner \begin{array}{l} [\texttt{cntr}\ \leftarrow \texttt{cntr}\ - 1] \\ [\ulcorner \mathcal{E}^{src}_1 \urcorner^{spec} \leftarrow \texttt{st(cntr)}\ ] \end{array}\ , \psi^{bc}_{exc})$$

{ *as* $\ulcorner \psi^{postN} \urcorner$ *does not contain stack counter expressions from Property 2 on page 11* }

(2)

$$wp^{bc}(\ \ulcorner \mathcal{E}^{src}_2 \urcorner\ , \ulcorner \mathsf{nPost}^{src}\urcorner \begin{array}{l} [\texttt{cntr}\ \leftarrow \texttt{cntr}\ - 1] \\ [\ulcorner \mathcal{E}^{src}_1 \urcorner^{spec} \leftarrow \texttt{st(cntr)}\ ] \end{array}\ , \psi^{bc}_{exc})$$
$$=$$
$$wp^{bc}(\ \ulcorner \mathcal{E}^{src}_2 \urcorner\ , \ulcorner \mathsf{nPost}^{src}\urcorner[\ulcorner \mathcal{E}^{src}_1 \urcorner^{spec} \leftarrow \texttt{st(cntr + 1)}\ ], \psi^{bc}_{exc})$$

$\{$ *from Lemma 2 on page 16, (2) and (0)* $\}$

(4)

$\exists P, Q : \mathcal{F}^{src}$,
(4.1)
$wp^{src}(\ \mathcal{E}_1^{src} = \mathcal{E}_2^{src}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src})$

$= wp^{src}(\ \mathcal{E}_2^{src}\ , \mathsf{nPost}^{src}[\mathcal{E}_1^{src} \leftarrow \ulcorner \mathcal{E}_2^{src} \urcorner^{spec}], \mathsf{ePost}^{src})$
$= P \Rightarrow \mathsf{nPost}^{src}[\mathcal{E}_1^{src} \leftarrow \ulcorner \mathcal{E}_2^{src} \urcorner^{spec}] \wedge R$

(4.2)
$wp^{bc}(\ \ulcorner \mathcal{E}_1^{src} = \mathcal{E}_2^{src} \urcorner\ , \ulcorner \mathsf{nPost}^{src} \urcorner, \psi^{bc}_{exc})$
$=$
$wp^{bc}(\ \ulcorner \mathcal{E}_2^{src} \urcorner\ , \ulcorner \mathsf{nPost}^{src} \urcorner [\ulcorner \mathcal{E}_1^{src} \urcorner^{spec} \leftarrow], \psi^{bc}_{exc})$
$=$

$$\ulcorner P \urcorner \Rightarrow \ulcorner \mathcal{E}_1^{src} \urcorner^{spec} \begin{array}{l} [\ulcorner \mathcal{E}_1^{src} \urcorner^{spec} \leftarrow \mathtt{st(cntr\ )}\ ] \\ [\mathtt{cntr}\ \leftarrow \mathtt{cntr}\ + 1] \\ [\mathtt{st(\ cntr\ +1)}\ \leftarrow \ulcorner \mathcal{E}_2^{src} \urcorner^{spec}] \end{array}$$

$\wedge \ulcorner R \urcorner$
$\{$ *as there are no stack expressions in* $\ulcorner \mathcal{E}_1^{src} \urcorner^{spec}$ *and applying properties of substitution* $\}$
$=$
$\ulcorner P \urcorner \Rightarrow \ulcorner \mathsf{nPost}^{src} \urcorner^{spec} [\ulcorner \mathcal{E}_1^{src} \urcorner^{spec} \leftarrow \ulcorner \mathcal{E}_2^{src} \urcorner^{spec}]$

instance field assignment

$\{$ *by definition of the weakest precondition function for field assignment* $\}$

(1)

$wp^{src}(\ \mathcal{E}_1^{src}.f = \mathcal{E}_2^{src}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) =$

$$wp^{src}(\ \mathcal{E}_1^{src}\ ,\ \begin{array}{l} \mathbf{null} \neq eval(\mathcal{E}_1^{src}) \Rightarrow \mathsf{nPost}^{src}[f \leftarrow f \oplus [eval(\mathcal{E}_1^{src}) \rightarrow eval(\mathcal{E}_2^{src})]] \\ \wedge \\ \mathbf{null} = eval(\mathcal{E}_1^{src}) \Rightarrow \mathsf{ePost}^{src}(\ \mathtt{NullPointerExc}\ , \mathcal{E}_1^{src}.f = \mathcal{E}_2^{src}) \end{array}\ , \mathsf{ePost}^{src})$$

$\{$ *by the definition of the compiler function* $\}$

$wp^{bc}(\ \ulcorner \mathcal{E}_1^{src}.f = \mathcal{E}_2^{src} \urcorner\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src})$
$=$

$$wp^{bc}(\ \begin{array}{l} \ulcorner \mathcal{E}_1^{src} \urcorner; \\ \ulcorner \mathcal{E}_2^{src} \urcorner; \\ \mathtt{putfield}\ \ f; \end{array}\ , \ulcorner \mathsf{nPost}^{src} \urcorner, \psi^{bc}_{exc})$$

$\{$ *by the definition of the weakest precondition for* `putfield` $\}$

$=$

$wp^{bc}(\ \begin{array}{l}\ulcorner\mathcal{E}_1^{src}\urcorner;\\ \ulcorner\mathcal{E}_2^{src}\urcorner;\end{array}\ ,$

$\quad \mathbf{null} = \mathtt{st(cntr\ -1)} \Rightarrow \ulcorner\mathsf{nPost}^{src}\urcorner \begin{array}{l}[\mathtt{cntr} \leftarrow \mathtt{cntr}\ -2]\\ [f \leftarrow f \oplus [\mathtt{st(cntr\ -1\ )} \rightarrow \mathtt{st(cntr\ )}\ ]]\end{array}$

$\quad \wedge$

$\quad \mathbf{null} = \mathtt{st(cntr\ -1)} \Rightarrow \psi^{bc}_{exc}(\ \mathtt{NullPointerExc}\ ,\quad \mathtt{putfield}\quad \mathrm{f}\ ),$

$\quad \psi^{bc}_{exc})$

$\{$ $\ulcorner\mathsf{nPost}^{src}\urcorner$ *does not contain stack counter expressions from Property 2 on page 11* $\}$

$=$

$wp^{bc}(\ \begin{array}{l}\ulcorner\mathcal{E}_1^{src}\urcorner;\\ \ulcorner\mathcal{E}_2^{src}\urcorner;\end{array}\ ,$

$\quad \mathbf{null} = \mathtt{st(cntr\ -1)} \Rightarrow \ulcorner\mathsf{nPost}^{src}\urcorner[f \leftarrow f \oplus [\mathtt{st(cntr\ -1\ )} \rightarrow \mathtt{st(cntr\ )}\ ]]$

$\quad \wedge$

$\quad \mathbf{null} = \mathtt{st(cntr\ -1)} \Rightarrow \psi^{bc}_{exc}(\ \mathtt{NullPointerExc}\ ,\quad \mathtt{putfield}\quad \mathrm{f}\ ),$

$\quad \psi^{bc}_{exc})$

$\{$ *by definition of the weakest precondition function for a sequence of bytecode instructions* $\}$

(2)

$=$

$wp^{bc}(\ \ulcorner\mathcal{E}_1^{src}\urcorner\ ,$

$\quad wp^{bc}(\ \ulcorner\mathcal{E}_2^{src}\urcorner\ ,$

$\quad\quad \mathbf{null} \neq \mathtt{st(cntr\ -1)} \Rightarrow \ulcorner\mathsf{nPost}^{src}\urcorner[f \leftarrow f \oplus [\mathtt{st(\ cntr\ -\ 1\ )}\ ] \rightarrow \mathtt{st(cntr\ )}\ ]$

$\quad\quad \wedge$

$\quad\quad \mathbf{null} = \mathtt{st(cntr\ -1)} \Rightarrow \psi^{bc}_{exc}(\ \mathtt{NullPointerExc}\ ,\quad \mathtt{putfield}\quad \mathrm{f}\ ),$

$\quad\quad \psi^{bc}_{exc}),$

$\quad \psi^{bc}_{exc})$

$\{$ *applying twice the lemma 2 on page 16* $\}$

(3)

$=$

$wp^{bc}(\ \ulcorner\mathcal{E}_1^{src}\urcorner\ ,$

$\quad wp^{bc}(\ \ulcorner\mathcal{E}_2^{src}\urcorner\ ,$

$\quad\quad \mathbf{null} \neq \mathtt{st(cntr\ -1)} \Rightarrow \ulcorner\mathsf{nPost}^{src}\urcorner[f \leftarrow f \oplus [\mathtt{st(\ cntr\ -\ 1\ )}\ ] \rightarrow \mathtt{st(cntr\ )}\ ]$

$\quad\quad \wedge$

$\quad\quad \mathbf{null} = \mathtt{st(cntr\ -1)} \Rightarrow \psi^{bc}_{exc}(\ \mathtt{NullPointerExc}\ ,\quad \mathtt{putfield}\quad \mathrm{f}\ )$

$\quad\quad \wedge$

$\quad\quad \mathtt{st(cntr\ )} = \ulcorner eval(\mathcal{E}_2^{src})\urcorner,$

$\quad\quad \psi^{bc}_{exc})$

$\quad \wedge$

$\quad \mathtt{st(cntr\ )} = \ulcorner eval(\mathcal{E}_1^{src})\urcorner,$

$\quad \psi^{bc}_{exc})$

$\{$ *from lemma* **??** *on page* **??** $\}$

(4)

$$
\begin{array}{l}
= \\
wp^{bc}(\ulcorner \mathcal{E}_1^{src} \urcorner, \\
\quad wp^{bc}(\ulcorner \mathcal{E}_2^{src} \urcorner, \\
\qquad \mathbf{null} \neq \texttt{st(cntr -1)} \Rightarrow \ulcorner \mathsf{nPost}^{src} \urcorner [f \leftarrow f \oplus [\texttt{st(cntr - 1)}] \rightarrow \texttt{st(cntr)}] \\
\qquad \wedge \\
\qquad \mathbf{null} = \texttt{st(cntr -1)} \Rightarrow \psi_{exc}^{bc}(\texttt{NullPointerExc}, \quad \texttt{putfield} \quad \mathrm{f}) \\
\qquad \wedge \\
\qquad \texttt{st(cntr)} = \ulcorner eval(\mathcal{E}_2^{src}) \urcorner \\
\qquad \wedge \\
\qquad \texttt{st(cntr - 1)} = \ulcorner eval(\mathcal{E}_1^{src}) \urcorner, \\
\qquad \psi_{exc}^{bc}) \\
, \\
\quad \psi_{exc}^{bc})
\end{array}
$$

$$
\begin{array}{l}
= \\
wp^{bc}(\ulcorner \mathcal{E}_1^{src} \urcorner, \\
\quad wp^{bc}(\ulcorner \mathcal{E}_2^{src} \urcorner, \\
\qquad \mathbf{null} \neq \ulcorner eval(\mathcal{E}_1^{src}) \urcorner \Rightarrow \ulcorner \mathsf{nPost}^{src} \urcorner [f \leftarrow f \oplus [\ulcorner eval(\mathcal{E}_1^{src}) \urcorner \rightarrow \ulcorner eval(\mathcal{E}_2^{src}) \urcorner]] \\
\qquad \wedge \\
\qquad \mathbf{null} = \ulcorner eval(\mathcal{E}_1^{src}) \urcorner \Rightarrow \psi_{exc}^{bc}(\texttt{NullPointerExc}, \quad \texttt{putfield} \quad \mathrm{f}) \\
, \\
\qquad \psi_{exc}^{bc}), \\
\quad \psi_{exc}^{bc})
\end{array}
$$

$\{$ *from (1) and (4) applying the induction hypothesis we obtain that the theorem holds in the case of instance field assignment* $\}$

field access

arithmetic expressions

$\{$ *by definition of the weakest precondition function for arithmetic expressions* $\}$

$$
\begin{array}{l}
wp^{src}(\mathcal{E}_1^{src}\ op\ \mathcal{E}_2^{src}, \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) = \\
wp^{src}(\mathcal{E}_1^{src}, wp^{src}(\mathcal{E}_2^{src}, \mathsf{nPost}^{src}, \mathsf{ePost}^{src}), \mathsf{ePost}^{src})
\end{array}
$$

$\{$ *by defintition of the compiler in section 4* $\}$

(0)

$$
\begin{array}{l}
wp^{bc}(\ulcorner \mathcal{E}_1^{src}\ op\ \mathcal{E}_2^{src} \urcorner, \ulcorner \mathsf{nPost}^{src} \urcorner, \ulcorner \mathsf{ePost}^{src} \urcorner) \\
= \\
wp^{bc}(\begin{array}{c} \ulcorner \mathcal{E}_1^{src} \urcorner; \\ \ulcorner \mathcal{E}_2^{src} \urcorner; \\ \mathsf{op} \end{array}, \\
\quad \ulcorner \mathsf{nPost}^{src} \urcorner, \\
\quad \psi_{exc}^{bc})
\end{array}
$$

{ *from the definition of the weakest precondition of the* `op` *instruction* }

(1)

$$= wp^{bc}\left(\begin{array}{l} \ulcorner \mathcal{E}_1^{src} \urcorner; \\ \ulcorner \mathcal{E}_2^{src} \urcorner; \end{array} , \begin{array}{l} \ulcorner \mathsf{nPost}^{src} \urcorner \begin{array}{l} [\mathtt{cntr} \leftarrow \mathtt{cntr} - 1] \\ [\mathtt{st(cntr - 1)} \leftarrow \mathtt{st(cntr)}\ op\ \mathtt{st(cntr - 1)}] \end{array}, \\ \ulcorner \mathsf{ePost}^{src} \urcorner \end{array} \right)$$

{ *the formula* $\ulcorner \psi \urcorner$ *does not contain* `cntr` *and* `st( cntr )` *expressions from Property 2 on page 11* }

(2)

$$= wp^{bc}\left(\begin{array}{l} \ulcorner \mathcal{E}_1^{src} \urcorner; \\ \ulcorner \mathcal{E}_2^{src} \urcorner; \end{array} , \begin{array}{l} \ulcorner \mathsf{nPost}^{src} \urcorner, \\ \mathsf{ePost}^{src} \end{array} \right)$$

{ *from (2), as* $\ulcorner \mathcal{E}_1^{src} \urcorner$ *and* $\ulcorner \mathcal{E}_2^{src} \urcorner$ *execute sequentially* }

(3)

$$= wp^{bc}(\ulcorner \mathcal{E}_1^{src} \urcorner , wp^{bc}(\ulcorner \mathcal{E}_2^{src} \urcorner , \ulcorner \mathsf{nPost}^{src} \urcorner, \psi^{bc}_{exc}), \psi^{bc}_{exc})$$

{ *by induction hypothesis over the structure of the source statements* }

(4)

$$wp^{src}(\mathcal{E}_2^{src} , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) =^{mod\ Names\ and\ bools} wp^{bc}(\ulcorner \mathcal{E}_2^{src} \urcorner , \ulcorner \mathsf{nPost}^{src} \urcorner, \psi^{bc}_{exc})$$

{ *by induction hypothesis over the structure of the source statements and (5)* }

(5)

$$\begin{array}{l} wp^{src}(\mathcal{E}_1^{src} , wp^{src}(\mathcal{E}_2^{src} , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}), \mathsf{ePost}^{src}) \\ =^{mod\ Names\ and\ bools} \\ wp^{bc}(\ulcorner \mathcal{E}_1^{src} \urcorner , \\ \quad wp^{bc}(\ulcorner \mathcal{E}_2^{src} \urcorner , \\ \qquad \ulcorner \mathsf{nPost}^{src} \urcorner, \\ \qquad \psi^{bc}_{exc}), \\ \quad \psi^{bc}_{exc}) \end{array}$$

{ *from (0), (3) and (5) the property holds in the case of arithmetic expression* }

cast expressions

{ *by definition of the weakest precondition function for cast expressions* }

$$wp^{src}(\ (\ \texttt{Class}\ )\ \mathcal{E}^{src}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) =$$
$$wp^{src}(\ \mathcal{E}^{src}\ ,\ \begin{array}{l} \backslash \textbf{\texttt{typeof(}}\mathcal{E}^{src}\textbf{\texttt{)}} <: \textbf{\texttt{Class}} \Rightarrow \\ \qquad wp^{src}(\ \mathcal{E}^{src}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) \\ \wedge \\ \neg \backslash \textbf{\texttt{typeof(}}\mathcal{E}^{src}\textbf{\texttt{)}} <: \textbf{\texttt{Class}} \Rightarrow \\ \qquad \mathsf{ePost}^{src}(\ \texttt{ClassCastException}\ ,\ \texttt{Class}\ \ \mathcal{E}^{src}) \end{array}, \mathsf{ePost}^{src})$$

{ *by the definition of the compiler* }

$$\begin{array}{l} wp^{bc}(\ \ulcorner(\ \texttt{Class}\ )\ \mathcal{E}^{src}\urcorner\ , \ulcorner\mathsf{nPost}^{src}\urcorner, \psi^{bc}_{exc}) \\ = \\ wp^{bc}(\ \begin{array}{l} \ulcorner\mathcal{E}^{src}\urcorner; \\ \texttt{checkCast}\quad \texttt{Class}\ ; \end{array}, \\ \quad \ulcorner\mathsf{nPost}^{src}\urcorner, \\ \quad \psi^{bc}_{exc}) \end{array}$$

{ *from the definition of the weakest precondition function for* `checkCast` }

(1)

$$\begin{array}{l} wp^{bc}(\ \ulcorner\mathcal{E}^{src}\urcorner\ , \\ \quad \begin{array}{l} \backslash \textbf{\texttt{typeof(}}\texttt{st(cntr)}\textbf{\texttt{)}} <: \texttt{Class}\ \Rightarrow \\ \qquad \ulcorner\mathsf{nPost}^{src}\urcorner \\ \wedge \\ \neg(\backslash \textbf{\texttt{typeof(}}\texttt{st(cntr)}\textbf{\texttt{)}} <: \texttt{Class}\ ) \Rightarrow \\ \qquad \psi^{bc}_{exc}(\texttt{ClassCastException},\quad \texttt{checkCast}\ )) \end{array}, \\ \quad \psi^{bc}_{exc}) \end{array}$$

{ *from lemma 2 on page 16 and (1)* }

(2)

$$\begin{array}{l} = \\ wp^{bc}(\ \ulcorner\mathcal{E}^{src}\urcorner\ , \\ \quad \begin{array}{l} \backslash \textbf{\texttt{typeof(}}\texttt{st(cntr)}\textbf{\texttt{)}} <: \texttt{Class}\ \Rightarrow \\ \qquad \ulcorner\mathsf{nPost}^{src}\urcorner \\ \wedge \\ \neg(\backslash \textbf{\texttt{typeof(}}\texttt{st(cntr)}\textbf{\texttt{)}} <: \texttt{Class}\ ) \Rightarrow \\ \qquad \psi^{bc}_{exc}(\texttt{ClassCastException},\quad \texttt{checkCast}\ )) \\ \wedge \\ \texttt{st(cntr)}\ = \ulcorner eval(\mathcal{E}^{src}_1)\urcorner \end{array}, \\ \quad \psi^{bc}_{exc}) \end{array}$$

(3)

$=$
$wp^{bc}(\ulcorner \mathcal{E}^{src} \urcorner ,$
$\quad \backslash \textit{typeof}(\ulcorner eval(\mathcal{E}_1^{src})\urcorner) <: \texttt{Class} \Rightarrow$
$\qquad \ulcorner \mathsf{nPost}^{src} \urcorner$
$\quad \wedge$,
$\quad \neg(\backslash \textit{typeof}(\ulcorner eval(\mathcal{E}_1^{src})\urcorner) <: \texttt{Class} \ ) \Rightarrow$
$\qquad \psi^{bc}_{exc}(\texttt{ClassCastException}, \quad \texttt{checkCast} \ ))$
$\quad \psi^{bc}_{exc})$

{ *from the induction hypothesis* }

$wp^{src}(\ \mathcal{E}^{src} \ ,$
$\backslash \textit{typeof}(\mathcal{E}^{src}) <: \texttt{Class} \Rightarrow$
$\qquad wp^{src}(\ \mathcal{E}^{src} \ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src})$
$\wedge$,
$\neg \ \backslash \textit{typeof}(\mathcal{E}^{src}) <: \texttt{Class} \Rightarrow$
$\qquad \mathsf{ePost}^{src}(\ \texttt{ClassCastException}, \quad \texttt{checkCast} \ )$
$\mathsf{ePost}^{src})$
$=^{mod\ Names\ and\ bools}$
$wp^{bc}(\ulcorner \mathcal{E}^{src} \urcorner ,$
$\quad \backslash \textit{typeof}(\texttt{st}(\texttt{cntr}\ )\ ) <: \textit{Class} \Rightarrow$
$\qquad \ulcorner \mathsf{nPost}^{src} \urcorner$
$\quad \wedge$,
$\quad \neg(\backslash \textit{typeof}(\ulcorner eval(\mathcal{E}^{src})\urcorner) <: \textit{Class} \ ) \Rightarrow$
$\qquad \psi^{bc}_{exc}(\ \texttt{ClassCastException}\ , \quad \texttt{checkCast} \quad)$
$\quad \psi^{bc}_{exc})$

{ *we can conclude that this case holds* }

Statements

compositional statements

$$wp^{src}(\ \mathcal{STMT}_1;\mathcal{STMT}_2 \ , \psi, \mathsf{ePost}^{src}) =$$

{ *by defintition* }

$$wp^{src}(\ \mathcal{STMT}_1 \ , wp^{src}(\ \mathcal{STMT}_2 \ , \psi, \mathsf{ePost}^{src}), \mathsf{ePost}^{src}) =$$

{ *applying the induction hypothesis* }

- $wp^{bc}(\ulcorner \mathcal{STMT}_2 \urcorner , \ulcorner \psi \urcorner, \psi^{bc}_{exc})$ *does not contain stack expressions*
-

(1) $wp^{src}(\ \mathcal{STMT}_2 \ , \psi, \mathsf{ePost}^{src}) =^{mod\ Names\ and\ bools} wp^{bc}(\ulcorner \mathcal{STMT}_2 \urcorner , \ulcorner \psi \urcorner, \psi^{bc}_{exc})$

{ *applying the induction hypothesis and from (1) we conclude* }

(2)

$wp^{src}(\ \mathcal{STMT}_1 \ , wp^{src}(\ \mathcal{STMT}_2 \ , \psi, \mathsf{ePost}^{src}), \mathsf{ePost}^{src})$
$=^{mod\ Names\ and\ bools}$
$wp^{bc}(\ulcorner \mathcal{STMT}_1 \urcorner , wp^{bc}(\ulcorner \mathcal{STMT}_2 \urcorner , \ulcorner \psi \urcorner, \psi^{bc}_{exc}), \ulcorner \mathsf{ePost}^{src} \urcorner)$

conditional statement

We suppose here that the condition of the statement is a boolean variable or constant (the case when the condition is a relation expression is similar)

$$wp^{src}(\ \texttt{if}\ (\mathcal{E}^{src})\ \texttt{then}\ \{\mathcal{STMT}_1\}\ \texttt{else}\ \{\mathcal{STMT}_2\}\ , \psi, \mathsf{ePost}^{src}) =$$

$\{$ *by definition* $\}$

$$(0) wp^{src}(\ \mathcal{E}^{src}\ ,\ \begin{array}{l} eval(\mathcal{E}^{src}) = \mathbf{true} \Rightarrow wp^{src}(\ \mathcal{STMT}_1\ , \mathsf{nPost}^{src}, \psi^{bc}_{exc}) \\ \wedge \\ eval(\mathcal{E}^{src}) = \mathbf{false} \Rightarrow wp^{src}(\ \mathcal{STMT}_2\ , \mathsf{nPost}^{src}, \psi^{bc}_{exc}) \end{array}\ , \mathsf{ePost}^{src})$$

$\{$ *by induction hypothesis* $\}$

(1)

- $wp^{bc}(\ \ulcorner \mathcal{STMT}_1 \urcorner\ , \ulcorner \psi \urcorner, \psi^{bc}_{exc})$ *does not contain stack expressions*
- $$\begin{array}{l} wp^{src}(\ \mathcal{STMT}_1\ , \mathsf{nPost}^{src}, \psi^{bc}_{exc}) \\ =^{mod\ Names\ and\ bools} \\ wp^{bc}(\ \ulcorner \mathcal{STMT}_1 \urcorner\ , \ulcorner \mathsf{nPost}^{src} \urcorner, \ulcorner \psi^{bc}_{exc} \urcorner) \end{array}$$

$\{$ *from (1)* $\}$

(1.1)

$$\begin{array}{l} wp^{src}(\ \mathcal{STMT}_1\ , \mathsf{nPost}^{src}, \psi^{bc}_{exc}) \\ =^{mod\ Names\ and\ bools} \\ wp^{bc}(\ \ulcorner \mathcal{STMT}_1 \urcorner\ , \ulcorner \mathsf{nPost}^{src} \urcorner, \ulcorner \psi^{bc}_{exc} \urcorner)[\texttt{cntr}\ \leftarrow \texttt{cntr}\ - 1] \end{array}$$

$\{$ *from (1.1)* $\}$

(1.2)

$$\begin{array}{l} eval(\mathcal{E}^{src}) = \mathbf{true} \Rightarrow wp^{src}(\ \mathcal{STMT}_1\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) \\ =^{mod\ Names\ and\ bools} \\ \ulcorner eval(\mathcal{E}^{src}) \urcorner = \ulcorner \mathbf{true} \urcorner \Rightarrow wp^{bc}(\ \ulcorner \mathcal{STMT}_1 \urcorner\ , \ulcorner \mathsf{nPost}^{src} \urcorner, \psi^{bc}_{exc})[\texttt{cntr}\ \leftarrow \texttt{cntr}\ - 1] \end{array}$$

$\{$ *by induction hypothesis* $\}$

(2)

$$\begin{array}{l} wp^{src}(\ \mathcal{STMT}_2\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) \\ =^{mod\ Names\ and\ bools} \\ wp^{bc}(\ \ulcorner \mathcal{STMT}_2 \urcorner\ , \ulcorner \mathsf{nPost}^{src} \urcorner, \psi^{bc}_{exc}) \end{array}$$

$\{$ *from lemme 1 on page 16* $\}$

(2.1)

$$\begin{array}{l} wp^{src}(\ \mathcal{STMT}_2\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) \\ = \\ wp^{bc}(\ \ulcorner \mathcal{STMT}_2; \urcorner\quad \texttt{goto}\quad l\ , \ulcorner \mathsf{nPost}^{src} \urcorner, \psi^{bc}_{exc}) \end{array}$$

{ *from (2.1) as* $\mathrm{wp}^{bc}(\ulcorner \mathcal{STMT}_2 \urcorner;\ \ \texttt{goto}\ \ l, \ulcorner \mathsf{nPost}^{src} \urcorner,)$ *does not contain the* `cntr` *expression from Property 2 on page 11* }

(2.2)

$$
\begin{array}{l}
wp^{src}(\ \mathcal{STMT}_2\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) \\
= \\
wp^{bc}(\ \ulcorner \mathcal{STMT}_2; \urcorner\ \texttt{goto}\ \ l\ , \ulcorner \mathsf{nPost}^{src} \urcorner, \psi^{bc}_{exc}) \\
\qquad\qquad\qquad\qquad [\texttt{cntr} \leftarrow \texttt{cntr}\ - 1]
\end{array}
$$

(2.3)

$$
\begin{array}{l}
eval(\mathcal{E}^{src}) = \mathbf{false} \Rightarrow wp^{src}(\ \mathcal{STMT}_2\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) \\
=^{mod\ Names\ and\ bools} \\
\ulcorner eval(\mathcal{E}^{src}) \urcorner = \ulcorner \mathbf{false} \urcorner \Rightarrow wp^{bc}(\ \ulcorner \mathcal{STMT}_2 \urcorner\ \ \texttt{goto}\ \ \ l\ , \ulcorner \mathsf{nPost}^{src} \urcorner, \psi^{bc}_{exc}) \\
\qquad\qquad\qquad\qquad\qquad [\texttt{cntr} \leftarrow \texttt{cntr}\ - 1]
\end{array}
$$

{ *from (2.3) and (1.2)* }

(3)

$$
\begin{array}{l}
eval(\mathcal{E}^{src}) = \mathbf{false} \Rightarrow wp^{src}(\ \mathcal{STMT}_2\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) \\
\wedge \\
eval(\mathcal{E}^{src}) = \mathbf{true} \Rightarrow wp^{src}(\ \mathcal{STMT}_1\ , \mathsf{nPost}^{src}, \psi^{bc}_{exc}) \\
\\
=^{mod\ Names\ and\ bools} \\
\\
\ulcorner eval(\mathcal{E}^{src}) \urcorner = \ulcorner \mathbf{false} \urcorner \Rightarrow wp^{bc}(\ \ulcorner \mathcal{STMT}_2 \urcorner\ \ \texttt{goto}\ \ \ l\ , \ulcorner \mathsf{nPost}^{src} \urcorner, \ulcorner \psi^{bc}_{exc} \urcorner) \\
\qquad\qquad\qquad\qquad\qquad [\texttt{cntr} \leftarrow \texttt{cntr}\ - 1] \\
\wedge \\
\ulcorner eval(\mathcal{E}^{src}) \urcorner = \ulcorner \mathbf{true} \urcorner \Rightarrow wp^{bc}(\ \ulcorner \mathcal{STMT}_1 \urcorner\ , \ulcorner \mathsf{nPost}^{src} \urcorner, \ulcorner \psi^{bc}_{exc} \urcorner) \\
\qquad\qquad\qquad\qquad\qquad [\texttt{cntr} \leftarrow \texttt{cntr}\ - 1]
\end{array}
$$

{ *from the definitions of the predicate transformer for the bytecode instruction* `if_cond` }

(4)

$$
wp^{bc}\left(\ \begin{array}{l}
\ulcorner \mathcal{E}^{src} \urcorner; \\
\quad \texttt{if\_cond}\ \ \texttt{l}\ \ ; \\
\ulcorner \mathcal{STMT}_2 \urcorner; \\
goto\ l; \\
\ulcorner \mathcal{STMT}1 \urcorner; \\
l : \ldots
\end{array}\ , \ulcorner \mathsf{nPost}^{src} \urcorner, \psi^{bc}_{exc}\right)
$$

$$=$$

$$
\begin{array}{l}
wp^{bc}(\ \ulcorner \mathcal{E}^{src} \urcorner\ , \\
\qquad \ulcorner \texttt{st(cntr)}\ \urcorner = \ulcorner \mathbf{true} \urcorner \Rightarrow wp^{bc}(\ \ulcorner \mathcal{STMT}_1 \urcorner\ , \ulcorner \mathsf{nPost}^{src} \urcorner, \ulcorner \psi^{bc}_{exc} \urcorner) \\
\qquad\qquad\qquad\qquad\qquad\qquad [\texttt{cntr} \leftarrow \texttt{cntr}\ - 1] \\
\qquad \wedge \\
\qquad \ulcorner \texttt{st(cntr)}\ \urcorner = \ulcorner \mathbf{false} \urcorner \Rightarrow wp^{bc}(\ \ulcorner \mathcal{STMT}_2; \urcorner goto\ l\ , \ulcorner \mathsf{nPost}^{src} \urcorner, \ulcorner \psi^{bc}_{exc} \urcorner) \quad , \\
\qquad\qquad\qquad\qquad\qquad\qquad [\texttt{cntr} \leftarrow \texttt{cntr}\ - 1] \\
\quad \psi^{bc}_{exc})
\end{array}
$$

$\{$ *from (4) and lemma 2 on page 16* $\}$

(5)

$=$

$$
\begin{array}{l}
wp^{bc}(\ \ulcorner \mathcal{E}^{src} \urcorner\ , \\
\quad \ulcorner \texttt{st(cntr)}\ \urcorner = \ulcorner \textbf{true} \urcorner \Rightarrow wp^{bc}(\ \ulcorner \mathcal{STMT}_1 \urcorner\ , \ulcorner \mathsf{nPost}^{src} \urcorner, \psi^{bc}_{exc}) \\
\qquad\qquad\qquad\qquad [\texttt{cntr } \leftarrow \texttt{cntr } - 1] \\
\quad \wedge \\
\quad \ulcorner \texttt{st(cntr)}\ \urcorner = \ulcorner \textbf{false} \urcorner \Rightarrow wp^{bc}(\ \ulcorner \mathcal{STMT}_2 ; \urcorner goto\ l\ , \ulcorner \mathsf{nPost}^{src} \urcorner, \psi^{bc}_{exc})\ \ , \\
\qquad\qquad\qquad\qquad [\texttt{cntr } \leftarrow \texttt{cntr } - 1] \\
\quad \wedge \\
\quad \texttt{st(cntr)}\ = \ulcorner eval(\mathcal{E}^{src}) \urcorner \\
\quad \psi^{bc}_{exc})
\end{array}
$$

(6)

$=$

$$
\begin{array}{l}
wp^{bc}(\ \ulcorner \mathcal{E}^{src} \urcorner ;\ , \\
\quad \ulcorner eval(\mathcal{E}^{src}) \urcorner = \ulcorner \textbf{true} \urcorner \Rightarrow wp^{bc}(\ \ulcorner \mathcal{STMT}_1 \urcorner\ , \ulcorner \mathsf{nPost}^{src} \urcorner, \ulcorner \psi^{bc}_{exc} \urcorner) \\
\qquad\qquad\qquad\qquad [\texttt{cntr } \leftarrow \texttt{cntr } - 1] \\
\quad \wedge \\
\quad \ulcorner eval(\mathcal{E}^{src}) \urcorner = \ulcorner \textbf{false} \urcorner \Rightarrow wp^{bc}(\ \ulcorner \mathcal{STMT}_2 ; \urcorner goto\ l\ , \ulcorner \mathsf{nPost}^{src} \urcorner, \ulcorner \psi^{bc}_{exc} \urcorner)\ \ , \\
\qquad\qquad\qquad\qquad [\texttt{cntr } \leftarrow \texttt{cntr } - 1] \\
\quad \psi^{bc}_{exc})
\end{array}
$$

$\{$ *from (0), (6) and (3) applying the induction hypothesis* $\}$

$$
\begin{array}{l}
wp^{src}(\ \texttt{if}\ (\mathcal{E}^{src})\ \texttt{then}\ \{\mathcal{STMT}\}\ \texttt{else}\ \{\mathcal{STMT}\}\ , \mathsf{nPost}^{src}, \psi^{bc}_{exc}) \\
=^{mod\ Names\ and\ bools} \\
wp^{bc}(\ \ulcorner \texttt{if}\ (\mathcal{E}^{src})\ \texttt{then}\ \{\mathcal{STMT}\}\ \texttt{else}\ \{\mathcal{STMT}\} \urcorner\ , \ulcorner \mathsf{nPost}^{src} \urcorner, \ulcorner \psi^{bc}_{exc} \urcorner)
\end{array}
$$

try catch statement

$$wp^{src}(\ \texttt{try}\ \{\mathcal{STMT}_1\}\ \texttt{catch}(Class\ )\ \{\mathcal{STMT}_2\}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) =$$

$$wp^{src}(\ \mathcal{STMT}_1\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src} \oplus [Class, \mathcal{STMT}_1 \longrightarrow wp^{src}(\ \mathcal{STMT}_2\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src})])$$

$\{$ *We now show that for every exception the function* $\mathsf{ePost}^{src} \oplus [Class \longrightarrow \mathrm{wp}^{src}(\ \mathcal{STMT}_2\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src})](\mathcal{STMT}_1,\ \texttt{Exc})$ *is syntactically equivalent to* $\psi^{bc}_{exc}(\ulcorner \mathcal{STMT}_1 \urcorner,\ \texttt{Exc})$ *. By definition, if an exception of type* `Exc` *is thrown during the execution of* $\mathcal{STMT}_1$*, we have the following:* $\}$

(1)

$$\mathsf{ePost}^{src} \oplus [\mathtt{Class} \longrightarrow wp^{src}(\ \mathcal{STMT}_2\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src})](\mathcal{STMT}_1,\ \mathtt{Exc}) = \begin{cases} wp^{src}(\ \mathcal{STMT}_2\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) & if\ \mathtt{Exc} <: \mathtt{Class} \\ wp^{src}(\ \text{handler}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) & if\ \neg\ (\mathtt{Exc} <: \mathtt{Class})\ and \\ & \mathtt{Exc}\ is\ handled\ by\ \ \text{handler} \\ \mathsf{exc}^{src}_{\mathtt{m}}(\mathtt{Exc}) & else \end{cases}$$

{ *by definition, if an exception of type* Exc *is thrown during the execution of* $\ulcorner\mathcal{STMT}_1\urcorner$ *at instruction at index ind as we know that an exception handler* $\ulcorner\mathcal{STMT}_2\urcorner$ *for this exception exists* }

(2)

$$\psi^{bc}_{exc}(\mathtt{Exc}, ind) = \begin{cases} wp^{bc}(\ulcorner\mathcal{STMT}_2\urcorner, \ulcorner\mathsf{nPost}^{src}\urcorner, \psi^{bc}_{exc}) & if\ \mathtt{Exc} <: \mathtt{Class} \\ wp^{bc}(\ulcorner\ \text{handler}\urcorner, \ulcorner\mathsf{nPost}^{src}\urcorner, \psi^{bc}_{exc}) & if\ \neg(\mathtt{Exc} <: \mathtt{Class})\ and \\ & \mathtt{Exc}\ is\ handled\ by\ \ulcorner\ \text{handler}\urcorner \\ exc^{bc}_{\mathtt{m}}(\mathtt{Exc}) & else \end{cases}$$

{ *by induction hypothesis* }

(3)

$$wp^{src}(\ \mathcal{STMT}_2\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) =^{mod\ Names\ and\ bools} wp^{bc}(\ulcorner\mathcal{STMT}_2\urcorner, \ulcorner\mathsf{nPost}^{src}\urcorner, \psi^{bc}_{exc})$$

{ *from (1),(2) and (3)* }

(4)

$$\forall \mathtt{Exc}\ \ \mathsf{ePost}^{src} \oplus [\mathtt{Class} \longrightarrow wp^{src}(\ \mathcal{STMT}_2\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src})](\mathtt{Exc}, \mathcal{STMT}_1) =^{mod\ names} \psi^{bc}_{exc}(\mathtt{Exc}\ , \ulcorner\mathcal{STMT}_1\urcorner)$$

{ *from (4) and by induction hypothesis* }

(5)

$$wp^{src}(\ \mathcal{STMT}_1\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src} \oplus [Class, \mathcal{STMT}_1 \longrightarrow wp^{src}(\ \mathcal{STMT}_2\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src})]) =^{mod\ Names\ and\ bools} wp^{bc}\left(\begin{array}{l} \ulcorner\mathcal{STMT}_1\urcorner \\ goto\ l; \\ ... \\ l: \end{array} , \ulcorner\mathsf{nPost}^{src}\urcorner, \psi^{bc}_{exc}\right)$$

{ *from (5) and the definition of the weakest precondition* }

$wp^{src}($ `try`$\{\mathcal{STMT}_1\}$`catch( Class)`$\{\mathcal{STMT}_2\}$ $, \mathsf{nPost}^{src}, \mathsf{ePost}^{src})$
$=^{mod\ Names\ and\ bools}$

$$wp^{bc}\left(\begin{array}{l} \ulcorner \mathcal{STMT}_1 \urcorner \\ goto\ l; \\ \ulcorner \mathcal{STMT}_2 \urcorner \\ goto\ l; \\ ... \\ l: \end{array} , \ulcorner \mathsf{nPost}^{src} \urcorner, \psi^{bc}_{exc} \right)$$

try finally statement

{ *by definition* }

$wp^{src}($ `try`$\{\mathcal{STMT}_1\}$ `finally` $\{\mathcal{STMT}_2\}$ $, \psi^{postN}, \mathsf{ePost}^{src})$

$=$

$$wp^{src}(\ \mathcal{STMT}_1\ , wp^{src}(\ \mathcal{STMT}_2\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}), \\ \mathsf{ePost}^{src} \oplus [Exception \longrightarrow wp^{src}(\ \mathcal{STMT}_2\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src})])$$

Note: exc is the exception object thrown by $\mathcal{STMT}_2$

{ *from the definition of the weakest precondition and the definition of the compiler function for* `try` $\{\mathcal{STMT}\}$ `finally` $\{\mathcal{STMT}\}$ *statements* }

$wp^{bc}(\ \ulcorner$`try`$\{\mathcal{STMT}_1\}$ `finally` $\{\mathcal{STMT}_2\}\urcorner\ , \ulcorner \psi^{postN} \urcorner, \psi^{bc}_{exc})$
$=$

$$wp^{bc}\left(\begin{array}{l} \ulcorner \mathcal{STMT}_1 \urcorner; \\ \ \mathtt{jsr}\ \ s; \\ \ \mathtt{goto}\ \ \ l; \\ \\ h:\ \mathtt{astore}\ \ e; \\ \ \mathtt{jsr}\ \ s; \\ \ \mathtt{aload}\ \ e; \\ \ \mathtt{athrow}\ ; \\ \\ s:\ \mathtt{astore}\ \ k; \\ \ulcorner \mathcal{STMT}_2 \urcorner; \\ \ \mathtt{ret}\ \ k \\ l: \end{array} , \ulcorner \psi^{postN} \urcorner, \psi^{bc}_{exc} \right)$$

{ *from the definition of the weakest precondition* `goto` *instruction* }

$$= wp^{bc}\left(\begin{array}{l} \ulcorner \mathcal{STMT}_1 \urcorner; \\ \texttt{jsr} \;\; s; \\ s: \; \texttt{astore} \;\; k; \\ \ulcorner \mathcal{STMT}_2 \urcorner; \\ \texttt{ret} \;\; k \end{array}, \ulcorner \psi^{postN} \urcorner, \psi^{bc}_{exc} \right)$$

{ *from the definition of the weakest precondition* `jsr` *instruction* }

$$= wp^{bc}(\ulcorner \mathcal{STMT}_1 \urcorner , wp^{bc}\left(\begin{array}{l} s: \; \texttt{astore} \;\; k; \\ \ulcorner \mathcal{STMT}_2 \urcorner; \\ \texttt{ret} \;\; k \end{array}, \ulcorner \psi^{postN} \urcorner, \psi^{bc}_{exc}\right), \psi^{bc}_{exc})$$

{ *from the definition of the weakest precondition* `ret` *instruction* }

$$= wp^{bc}(\ulcorner \mathcal{STMT}_1 \urcorner , wp^{bc}\left(\begin{array}{l} s: \; \texttt{astore} \;\; k; \\ \ulcorner \mathcal{STMT}_2 \urcorner; \end{array}, \ulcorner \psi^{postN} \urcorner, \psi^{bc}_{exc}\right), \psi^{bc}_{exc})$$

{ *as the instructions* $s:$ `astore` $k; \ulcorner \mathcal{STMT}_2 \urcorner$ *execute sequentially* }

$$\begin{array}{l} = \\ wp^{bc}(\ulcorner \mathcal{STMT}_1 \urcorner , \\ \quad wp^{bc}(s: \; \texttt{astore} \;\; k , \\ \quad\quad wp^{bc}(\ulcorner \mathcal{STMT}_2 \urcorner , \\ \quad\quad \ulcorner \psi^{postN} \urcorner, \\ \quad\quad \psi^{bc}_{exc}), \\ \quad \psi^{bc}_{exc}), \\ \psi^{bc}_{exc}) \end{array}$$

{ *from the definition of the weakest precondition* `astore` *instruction* }

$$= wp^{bc}(\ulcorner \mathcal{STMT}_1 \urcorner , wp^{bc}(\ulcorner \mathcal{STMT}_2 \urcorner , \ulcorner \psi^{postN} \urcorner, \psi^{bc}_{exc}) \begin{array}{l} [\texttt{cntr} \leftarrow \texttt{cntr} - 1] \\ [\texttt{reg}_\texttt{k} \leftarrow \texttt{st(cntr)}] \end{array}, \psi^{bc}_{exc})$$

{ *by induction hypothesis there are no* `cntr` *expressions in* $\mathrm{wp}^{bc}(\ulcorner \mathcal{STMT}_2 \urcorner , \ulcorner \psi^{postN} \urcorner, \psi^{bc}_{exc})$ }

$$= wp^{bc}(\ulcorner \mathcal{STMT}_1 \urcorner , wp^{bc}(\ulcorner \mathcal{STMT}_2 \urcorner , \ulcorner \psi^{postN} \urcorner, \psi^{bc}_{exc})[\texttt{reg}_\texttt{k} \leftarrow \texttt{st(cntr)}], \psi^{\texttt{bc}}_{\texttt{exc}})$$

{ *there* $\texttt{reg}_\texttt{k}$ *does not appear in the specification, neither is used in* $\mathcal{STMT}_1$ *by hypothesis, see Section 4* }

$$= wp^{bc}(\ulcorner \mathcal{STMT}_1 \urcorner , wp^{bc}(\ulcorner \mathcal{STMT}_2 \urcorner , \ulcorner \psi^{postN} \urcorner, \psi^{bc}_{exc}), \psi^{bc}_{exc})$$

throw statement

{ *by definition* }

$$wp^{src}(\ \texttt{throw}\ \mathcal{E}^{src}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src})$$
$$=$$

(1)

$$wp^{src}(\ \mathcal{E}^{src}\ ,\ \begin{array}{l} eval(\mathcal{E}^{src}) \neq \mathbf{null} \Rightarrow \mathsf{ePost}^{src}(\backslash\textit{\texttt{typeof}}(eval(\mathcal{E}^{src})), \mathcal{E}^{src}) \\ \wedge \\ eval(\mathcal{E}^{src}) = \mathbf{null} \Rightarrow \mathsf{ePost}^{src}(\ \texttt{NullPointerExc}\ , \mathcal{E}^{src}) \end{array}\ , \mathsf{ePost}^{src})$$

{ *by definition of the compiler function* }

$$wp^{bc}(\ \ulcorner\texttt{throw}\ \mathcal{E}^{src}\urcorner\ , \ulcorner\mathsf{nPost}^{src}\urcorner, \psi^{bc}_{exc})$$
$$=$$
$$wp^{bc}(\ \begin{array}{c} \ulcorner\mathcal{E}^{src}\urcorner \\ \texttt{athrow} \end{array}\ ,\ \ulcorner\psi^{postN}\urcorner,\ \psi^{bc}_{exc})$$

{ *by definition of the function* wp^{bc} *for* `athrow` }

(2)

$$=$$
$$wp^{bc}(\ \ulcorner\mathcal{E}^{src}\urcorner\ ,\ \begin{array}{l} \texttt{st(cntr)}\ \neq \mathbf{null} \Rightarrow \\ \quad \psi^{bc}_{exc}(\backslash\textit{\texttt{typeof}}(\mathcal{E}^{src}), \ulcorner\mathcal{E}^{src}\urcorner) \\ \wedge \quad \texttt{st(cntr)}\ = \mathbf{null} \Rightarrow \\ \quad \psi^{bc}_{exc}(\ \texttt{NullPointerExc}, \ulcorner\mathcal{E}^{src}\urcorner) \end{array}\ ,\ \psi^{bc}_{exc})$$

{ *by initial hypothesis* $\forall \texttt{Exc}, \mathcal{E}.\mathsf{ePost}^{src}(\texttt{Exc}, \mathcal{E}) =^{mod\ Names\ and\ bools} \psi^{bc}_{exc}(\texttt{Exc}, \ulcorner\mathcal{E}\urcorner)$ }

(3)

$$\begin{array}{l} eval(\mathcal{E}^{src}) \neq \mathbf{null} \Rightarrow \mathsf{ePost}^{src}(\backslash\textit{\texttt{typeof}}(eval(\mathcal{E}^{src})), \mathcal{E}^{src}) \\ \wedge eval(\mathcal{E}^{src}) = \mathbf{null} \Rightarrow \mathsf{ePost}^{src}(\ \texttt{NullPointerExc}\ , \mathcal{E}^{src}) \\ =^{mod\ Names\ and\ bools} \\ \texttt{st(cntr)}\ \neq \mathbf{null} \Rightarrow \psi^{bc}_{exc}(\backslash\textit{\texttt{typeof}}(\mathcal{E}^{src}), \ulcorner\mathcal{E}^{src}\urcorner) \\ \wedge \\ \texttt{st(cntr)}\ = \mathbf{null} \Rightarrow \psi^{bc}_{exc}(\ \texttt{NullPointerExc}, \ulcorner\mathcal{E}^{src}\urcorner) \end{array}$$

{ *from (3) we can apply the induction hypothesis* }

$$(1) =^{mod\ Names\ and\ bools} (2)$$

{ *and we can conclude that the hypothesis hold* }

loop statement

{ *we name the loop invariant* INV }

$$wp^{src}(\ \texttt{while}\ (\mathcal{E}^{src})\ \ \{\mathcal{STMT}\}\ , \psi, \mathsf{ePost}^{src})$$

{ *by defintition* **??** }

(0)

$$=$$

$$\texttt{INV} \wedge$$

$$\forall\ m_i.i = 1..k$$

$$\quad \texttt{INV} \Rightarrow$$

$$wp^{src}(\ \mathcal{E}^{src}\ , \begin{array}{l} eval(\mathcal{E}^{src}) = \textbf{true} \Rightarrow \\ \quad wp^{src}(\ \mathcal{STMT}\ , \texttt{INV}, \mathsf{ePost}^{src}) \\ eval(\mathcal{E}^{src}) = \textbf{false} \Rightarrow \mathsf{nPost}^{src} \end{array} , \mathsf{ePost}^{src})$$

{ *by definition 4 of the compiler from source language to bytecode language* }

(1)

$$wp^{bc}(\ \ulcorner\texttt{while}\ (\mathcal{E}^{src})\ \ \{\mathcal{STMT}\}\urcorner\ , \ulcorner\psi\urcorner, \mathsf{ePost}^{src})$$

$$=$$

$$wp^{bc}(\ \begin{array}{l} \ \texttt{goto}\ \ loopEntry; \\ loopBody : \ulcorner\mathcal{STMT}\urcorner; \\ \ulcorner\texttt{INV}\urcorner; \\ loopEntry : \ulcorner\mathcal{E}^{src}\urcorner; \\ \ \texttt{if\_cond}\ \ loopBody; \end{array} , \ulcorner\psi\urcorner, \psi^{bc}_{exc})$$

{ }

(2)

Say why is it like this ?

$$=$$

$$wp^{bc}(\ \begin{array}{l} \ \texttt{goto}\ \ loopEntry; \\ \ulcorner\texttt{INV}\ \urcorner; \\ loopEntry : \ulcorner\mathcal{E}^{src}\urcorner; \end{array} ,$$

$$\begin{array}{l} \texttt{st(cntr)}\ == \ulcorner\textbf{true}\urcorner \Rightarrow wp^{bc}(\ \ulcorner\mathcal{STMT}\urcorner\ , \ulcorner\texttt{INV}\ \urcorner, \psi^{bc}_{exc}) \\ \wedge \\ \texttt{st(cntr)}\ == \ulcorner\textbf{false}\urcorner \Rightarrow \ulcorner\psi\urcorner \end{array} ,$$

$$\psi^{bc}_{exc})$$

{ *from lemma 2 on page 16 and from (2)* }

(3)

$$
= wp^{bc}\left(\begin{array}{l}\texttt{goto}\ \ loopEntry;\\ \ulcorner\texttt{INV}\urcorner;\\ loopEntry : \ulcorner\mathcal{E}^{src}\urcorner;\end{array},\ \begin{array}{l}\ulcorner\mathcal{E}^{src}\urcorner == \ulcorner\mathbf{true}\urcorner \Rightarrow wp^{bc}(\ulcorner\mathcal{STMT}\urcorner, \ulcorner\texttt{INV}\urcorner, \mathsf{ePost}^{src})\\ \wedge\\ \ulcorner\mathcal{E}^{src}\urcorner == \ulcorner\mathbf{false}\urcorner \Rightarrow \ulcorner\psi\urcorner\\ \wedge\texttt{st(cntr)} = \ulcorner eval(\mathcal{E}^{src})\urcorner\end{array},\ \psi^{bc}_{exc}\right)
$$

{ *by definition of the weakest precondition for loop entry instructions* }

$$
= wp^{bc}\left(\texttt{goto}\ \ loopEntry,\ \begin{array}{l}\ulcorner\texttt{INV}\urcorner \wedge\\ \forall\ m_i.i = 1..k\\ \ulcorner\texttt{INV}\urcorner \Rightarrow\\ \quad wp^{bc}\left(\ulcorner\mathcal{E}^{src}\urcorner,\ \begin{array}{l}\ulcorner\mathcal{E}^{src}\urcorner == \ulcorner\mathbf{true}\urcorner \Rightarrow\\ \quad wp^{bc}(\ulcorner\mathcal{STMT}\urcorner, \ulcorner\texttt{INV}\urcorner, \psi^{bc}_{exc})\\ \wedge\\ \ulcorner\mathcal{E}^{src}\urcorner == \ulcorner\mathbf{false}\urcorner \Rightarrow\\ \quad \ulcorner\psi\urcorner\end{array},\ \psi^{bc}_{exc}\right)\end{array},\ \psi^{bc}_{exc}\right)
$$

{ *by definition of the weakest precondition for* `goto` *instructions* }

(4)

$$
= \begin{array}{l}\ulcorner\texttt{INV}\urcorner \wedge\\ \forall\ m_i.i = 1..k\\ \ulcorner\texttt{INV}\urcorner \Rightarrow\\ \quad wp^{bc}\left(\ulcorner\mathcal{E}^{src}\urcorner,\ \begin{array}{l}\ulcorner\mathcal{E}^{src}\urcorner == \ulcorner\mathbf{true}\urcorner \Rightarrow\\ \quad wp^{bc}(\ulcorner\mathcal{STMT}\urcorner, \ulcorner\texttt{INV}\urcorner, \psi^{bc}_{exc})\\ \wedge\\ \ulcorner\mathcal{E}^{src}\urcorner == \ulcorner\mathbf{false}\urcorner \Rightarrow\\ \quad \ulcorner\psi\urcorner\end{array},\ \psi^{bc}_{exc}\right)\end{array}
$$

{ *from (0) and (4) applying the structural induction hypothesis we can conclude* }

$$
wp^{src}(\texttt{while}\ (\mathcal{E}^{src})\ \{\mathcal{STMT}\}, \psi, \mathsf{ePost}^{src}) = \textit{(0)}
$$

$$
=^{mod\ Names\ and\ bools}
$$

$$
wp^{bc}(\ulcorner\texttt{while}\ (\mathcal{E}^{src})\ \{\mathcal{STMT}\}\urcorner, \ulcorner\psi\urcorner, \psi^{bc}_{exc}) = \textit{(4)}
$$

Return statements *We consider only the case of a non void return*

{ *by definition of the weakest precondition for* `return` $\mathcal{E}^{src}$ }

(0)

$$wp^{src}(\ \mathtt{return}\ \mathcal{E}^{src}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src})$$
$$=$$
$$wp^{src}(\ \mathcal{E}^{src}\ , \mathsf{nPost}^{src}[\ \ \backslash\mathrm{result}\ \ \leftarrow eval(\mathcal{E}^{src})], \mathsf{ePost}^{src})$$

{ *by definition of the compiler in Section 4* }

$$wp^{bc}(\ \ulcorner \mathtt{return}\ \mathcal{E}^{src} \urcorner\ , \ulcorner \psi \urcorner, \psi^{bc}_{exc})$$

$$=$$

$$wp^{bc}(\ \begin{array}{c} \ulcorner \mathcal{E}^{src} \urcorner; \\ \mathtt{return} \end{array}\ , \ulcorner \psi \urcorner, \psi^{bc}_{exc})$$

{ *by definition of the weakest predicate transformer function for the* `return` *instruction* }

$$=$$

$$\begin{array}{l} wp^{bc}(\ \ulcorner \mathcal{E}^{src} \urcorner\ , \\ \quad \ulcorner \psi \urcorner[\ \ \backslash\mathrm{result}\ \ \leftarrow \mathtt{st(cntr\ )}\ ], \\ \quad \psi^{bc}_{exc}) \end{array}$$

{ *from lemma 2 on page 16* }

(1)

$$=$$

$$wp^{bc}(\ \ulcorner \mathcal{E}^{src} \urcorner\ ,\ \begin{array}{l} \ulcorner \psi \urcorner[\ \ \backslash\mathrm{result}\ \ \leftarrow \mathtt{st(cntr\ )}\ ] \\ \wedge \\ \mathtt{st(cntr\ )}\ \ = \ulcorner eval(\mathcal{E}^{src}) \urcorner \end{array}\ , \psi^{bc}_{exc})$$

(2)

$$=$$
$$\begin{array}{l} wp^{bc}(\ \ulcorner \mathcal{E}^{src} \urcorner\ , \\ \quad \ulcorner \psi \urcorner[\ \ \backslash\mathrm{result}\ \ \leftarrow \ulcorner eval(\mathcal{E}^{src}) \urcorner], \\ \quad \psi^{bc}_{exc}) \end{array}$$

{ *as* $\ulcorner \psi \urcorner[\ \ \backslash\mathrm{result}\ \ \leftarrow \ulcorner \mathrm{eval}(\mathcal{E}^{src}) \urcorner]$ *does not contain stack expressions, we can apply the induction hypothesis* }

(4)

$$wp^{src}(\ \mathcal{E}^{src}\ , \mathsf{nPost}^{src}[\quad \backslash\text{result} \ \leftarrow eval(\mathcal{E}^{src})], \mathsf{ePost}^{src})$$

$$=^{mod\ Names\ and\ bools}$$

$$\begin{array}{l} wp^{bc}(\ \ulcorner \mathcal{E}^{src} \urcorner\ , \\ \quad \ulcorner \psi \urcorner\ [\quad \backslash\text{result} \ \leftarrow \ulcorner eval(\mathcal{E}^{src}) \urcorner]\ , \\ \quad \psi^{bc}_{exc}) \end{array}$$

{ *from (0) and (3) and (4) this case holds* }

$$wp^{src}(\ \texttt{return}\ \mathcal{E}^{src}\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src})$$

$$=^{mod\ Names\ and\ bools}$$

$$wp^{bc}(\ \ulcorner \texttt{return}\ \mathcal{E}^{src} \urcorner\ , \ulcorner \psi \urcorner, \psi^{bc}_{exc})$$

Instance Creation expressions *We will consider only the case when the constructor takes one argument for reasons of readability. The general case is straightforward.*

{ *from the definition of the weakest precondition for instance creation in the source language in section* **??** }

(1)

$$\begin{array}{l} wp^{src}(\ \mathbf{new}\ Class(\mathcal{E}^{src})\ , \mathsf{nPost}^{src}, \mathsf{ePost}^{src}) \\ = \\ wp^{src}(\ \mathcal{E}^{src}\ , \\ \quad \psi^{pre}(ConsClass) \left\{ \begin{array}{l} [\mathbf{this} \leftarrow ref_{Class}] \\ [arg_1 \leftarrow eval(\mathcal{E}^{src})] \end{array} \right. \\ \quad \wedge \\ \quad \forall\ mod_i . i = 1..n \\ \quad \left\{ \begin{array}{l} \mathsf{nPost}^{src}(ConsClass) \left\{ \begin{array}{l} [\mathbf{this} \leftarrow ref_{Class}] \\ [arg_1 \leftarrow eval(\mathcal{E}^{src})] \end{array} \right. \Rightarrow \mathsf{nPost}^{src} \\ \wedge \\ exc^{bc}_{ConsClass}(\texttt{Exc}_\texttt{1}) \Rightarrow \mathsf{ePost}^{\mathsf{src}}(\texttt{Exc}_\texttt{1}, \texttt{newClass}(\mathcal{E}^{\mathsf{src}})) \\ \wedge \\ \ldots \\ \wedge \\ exc^{bc}_{ConsClass}(\texttt{Exc}_\texttt{s}) \Rightarrow \mathsf{ePost}^{\mathsf{src}}(\texttt{Exc}_\texttt{s}, \texttt{newClass}(\mathcal{E}^{\mathsf{src}})) \end{array} \right. , \\ \quad \mathsf{ePost}^{src})[\mathit{Heap} \leftarrow \mathit{Heap} \oplus [ref_{Class} \rightarrow \texttt{Obj}_{\texttt{Class}}]] \end{array}$$

{ *by definition of the compiler function* }

$$\begin{array}{l} wp^{bc}(\ \ulcorner \mathbf{new}\ Class(\mathcal{E}^{src}) \urcorner\ , \ulcorner \mathsf{nPost}^{src} \urcorner, \psi^{bc}_{exc}) \\ = \\ wp^{bc}(\ \begin{array}{l} \texttt{new}\ \ Class; \\ \texttt{dup}\ ; \\ \ulcorner \mathcal{E}^{src} \urcorner; \\ \texttt{invoke}\ \ ConsClass; \end{array} \qquad , \ulcorner \mathsf{nPost}^{src} \urcorner, \psi^{bc}_{exc}) \end{array}$$

$\{$ *apply the rule for method invocation* $\}$

$$
\begin{array}{l}
= \\
wp^{bc}(\begin{array}{l} \texttt{new}\ \ Class; \\ \texttt{dup}\ ; \\ \ulcorner \mathcal{E}^{src} \urcorner; \end{array} , \\
\quad \ulcorner \psi^{pre}(ConsClass) \urcorner^{spec} \left\{ \begin{array}{l} [\ulcorner \mathbf{this} \urcorner^{spec} \leftarrow \texttt{st(cntr - 1)}] \\ [\ulcorner arg_1 \urcorner^{spec} \leftarrow \texttt{st(cntr)}] \end{array} \right. \\
\quad \wedge \\
\quad \forall\ mod_i.i = 1..n \\
\quad \left\{ \begin{array}{l}
\ulcorner \mathsf{nPost}^{src}(ConsClass) \urcorner^{spec} \left\{ \begin{array}{l} [\ulcorner \mathbf{this} \urcorner^{spec} \leftarrow \texttt{st(cntr - 1)}] \\ [\ulcorner arg_1 \urcorner^{spec} \leftarrow \texttt{st(cntr)}] \end{array} \right. \Rightarrow \mathsf{nPost}^{src} \\
\wedge \\
exc^{bc}_{ConsClass}(\texttt{Exc}_1) \Rightarrow \mathsf{ePost}^{\mathsf{src}}(\texttt{Exc}_1, \texttt{newClass}(\mathcal{E}^{\mathsf{src}})) \\
\wedge \\
\ldots \\
\wedge \\
exc^{bc}_{ConsClass}(\texttt{Exc}_\mathsf{s}) \Rightarrow \mathsf{ePost}^{\mathsf{src}}(\texttt{Exc}_\mathsf{s}, \texttt{newClass}(\mathcal{E}^{\mathsf{src}}))
\end{array} \right. , \\
\quad \psi^{bc}_{exc})
\end{array}
$$

$\{$ *applying the weakest precondition rule for a sequential list of instructions* $\}$

$$
\begin{array}{l}
= \\
wp^{bc}(\begin{array}{l} \texttt{new}\ \ Class; \\ \texttt{dup}\ ; \end{array} , \\
\quad wp^{bc}(\ulcorner \mathcal{E}^{src} \urcorner , \\
\qquad \ulcorner \psi^{pre}(ConsClass) \urcorner^{spec} \left\{ \begin{array}{l} [\ulcorner \mathbf{this} \urcorner^{spec} \leftarrow \texttt{st(cntr - 1)}] \\ [\ulcorner arg_1 \urcorner^{spec} \leftarrow \texttt{st(cntr)}] \end{array} \right. \\
\qquad \wedge \\
\qquad \forall\ mod_i.i = 1..n \\
\qquad \left\{ \begin{array}{l}
\ulcorner \mathsf{nPost}^{src}(ConsClass) \urcorner^{spec} \left\{ \begin{array}{l} [\ulcorner \mathbf{this} \urcorner^{spec} \leftarrow \texttt{st(cntr - 1)}] \\ [\ulcorner arg_1 \urcorner^{spec} \leftarrow \texttt{st(cntr)}] \end{array} \right. \Rightarrow \mathsf{nPost}^{src} \\
\wedge \\
exc^{bc}_{ConsClass}(\texttt{Exc}_1) \Rightarrow \mathsf{ePost}^{\mathsf{src}}(\texttt{Exc}_1, \texttt{newClass}(\mathcal{E}^{\mathsf{src}})) \\
\wedge \\
\ldots \\
\wedge \\
exc^{bc}_{ConsClass}(\texttt{Exc}_\mathsf{s}) \Rightarrow \mathsf{ePost}^{\mathsf{src}}(\texttt{Exc}_\mathsf{s}, \texttt{newClass}(\mathcal{E}^{\mathsf{src}}))
\end{array} \right. , \\
\qquad \psi^{bc}_{exc}), \\
\quad \psi^{bc}_{exc})
\end{array}
$$

$\{$ *applying lemmas 2 on page 16,* **??** *on* **??** $\}$

$$
\begin{array}{l}
= \\
wp^{bc}(\begin{array}{l}\texttt{new}\ \ Class; \\ \texttt{dup}\ ;\end{array} , \\
\quad wp^{bc}(\ulcorner \mathcal{E}^{src} \urcorner , \\
\qquad \ulcorner \psi^{pre}(ConsClass) \urcorner^{spec} \left\{ \begin{array}{l} [\ulcorner \mathbf{this} \urcorner^{spec} \leftarrow \texttt{st(cntr - 1)}] \\ [\ulcorner arg_1 \urcorner^{spec} \leftarrow \texttt{st(cntr)}] \end{array} \right. \\
\qquad \wedge \\
\qquad \forall\ mod_i . i = 1..n \\
\qquad \left\{ \begin{array}{l}
\ulcorner \mathsf{nPost}^{src}(ConsClass) \urcorner^{spec} \left\{ \begin{array}{l} [\ulcorner \mathbf{this} \urcorner^{spec} \leftarrow \texttt{st(cntr - 1)}] \\ [\ulcorner arg_1 \urcorner^{spec} \leftarrow \texttt{st(cntr)}] \end{array} \right. \quad \Rightarrow \mathsf{nPost}^{src} \\
\wedge \\
exc^{bc}_{ConsClass}(\texttt{Exc}_1) \Rightarrow \mathsf{ePost}^{\mathsf{src}}(\texttt{Exc}_1, \texttt{newClass}(\mathcal{E}^{\mathsf{src}})) \\
\wedge \\
\ldots \\
\wedge \\
exc^{bc}_{ConsClass}(\texttt{Exc}_s) \Rightarrow \mathsf{ePost}^{\mathsf{src}}(\texttt{Exc}_s, \texttt{newClass}(\mathcal{E}^{\mathsf{src}}))
\end{array} \right. \\
\qquad \wedge \\
\qquad \texttt{st(cntr)} \ = \ulcorner eval(\mathcal{E}^{src}) \urcorner , \\
\qquad \psi^{bc}_{exc}) \\
\quad \wedge \\
\quad \texttt{st(cntr)} \ = \texttt{st(cntr -1)} \ , \\
\quad \psi^{bc}_{exc})
\end{array}
$$

$\{$ *applying the lemma* **??** *on page* **??** $\}$

$$
\begin{array}{l}
= \\
wp^{bc}(\begin{array}{l}\texttt{new}\ \ Class; \\ \texttt{dup}\ ;\end{array} , \\
\quad wp^{bc}(\ulcorner \mathcal{E}^{src} \urcorner , \\
\qquad \ulcorner \psi^{pre}(ConsClass) \urcorner^{spec} \left\{ \begin{array}{l} [\ulcorner \mathbf{this} \urcorner^{spec} \leftarrow \texttt{st(cntr - 1)}] \\ [\ulcorner arg_1 \urcorner^{spec} \leftarrow \texttt{st(cntr)}] \end{array} \right. \\
\qquad \wedge \\
\qquad \forall\ mod_i . i = 1..n \\
\qquad \left\{ \begin{array}{l}
\ulcorner \mathsf{nPost}^{src}(ConsClass) \urcorner^{spec} \left\{ \begin{array}{l} [\ulcorner \mathbf{this} \urcorner^{spec} \leftarrow \texttt{st(cntr - 1)}] \\ [\ulcorner arg_1 \urcorner^{spec} \leftarrow \texttt{st(cntr)}] \end{array} \right. \quad \Rightarrow \mathsf{nPost}^{src} \\
\wedge \\
exc^{bc}_{ConsClass}(\texttt{Exc}_1) \Rightarrow \mathsf{ePost}^{\mathsf{src}}(\texttt{Exc}_1, \texttt{newClass}(\mathcal{E}^{\mathsf{src}})) \\
\wedge \\
\ldots \\
\wedge \\
exc^{bc}_{ConsClass}(\texttt{Exc}_s) \Rightarrow \mathsf{ePost}^{\mathsf{src}}(\texttt{Exc}_s, \texttt{newClass}(\mathcal{E}^{\mathsf{src}}))
\end{array} \right. \\
\qquad \wedge \\
\qquad \texttt{st(cntr)} \ = \ulcorner eval(\mathcal{E}^{src}) \urcorner \\
\qquad \wedge \\
\qquad \texttt{st(cntr -1)} \ = \texttt{st(cntr -2)} \ , \\
\qquad \psi^{bc}_{exc}) \\
, \\
\quad \psi^{bc}_{exc})
\end{array}
$$

$\{$ *apply the weakest precondition calculus for sequential instructions and lemma* **??** *on page* **??** $\}$

$$
\begin{array}{l}
= \\
wp^{bc}(\ \texttt{new}\ \ Class\ , \\
\quad wp^{bc}(\ \texttt{dup}\ ;\ , \\
\quad\quad wp^{bc}(\ \ulcorner \mathcal{E}^{src} \urcorner\ , \\
\quad\quad\quad \ulcorner \psi^{pre}(ConsClass) \urcorner^{spec} \left\{ \begin{array}{l} [\ulcorner \textbf{this} \urcorner^{spec} \leftarrow \texttt{st(cntr - 1)}\ ] \\ [\ulcorner arg_1 \urcorner^{spec} \leftarrow \texttt{st(cntr)}\ ] \end{array} \right. \\
\quad\quad\quad \wedge \\
\quad\quad\quad \forall\ mod_i.i = 1..n \\
\quad\quad\quad \left\{ \begin{array}{l}
\ulcorner \mathsf{nPost}^{src}(ConsClass) \urcorner^{spec} \left\{ \begin{array}{l} [\ulcorner \textbf{this} \urcorner^{spec} \leftarrow \texttt{st(cntr - 1)}\ ] \\ [\ulcorner arg_1 \urcorner^{spec} \leftarrow \texttt{st(cntr)}\ ] \end{array} \right. \Rightarrow \mathsf{nPost}^{src} \\
\wedge \\
exc^{bc}_{ConsClass}(\texttt{Exc}_1) \Rightarrow \mathsf{ePost}^{\mathsf{src}}(\texttt{Exc}_1, \texttt{newClass}(\mathcal{E}^{\mathsf{src}})) \\
\wedge \\
\ldots \\
\wedge \\
exc^{bc}_{ConsClass}(\texttt{Exc}_\texttt{s}) \Rightarrow \mathsf{ePost}^{\mathsf{src}}(\texttt{Exc}_\texttt{s}, \texttt{newClass}(\mathcal{E}^{\mathsf{src}}))
\end{array} \right. \\
\quad\quad\quad \wedge \\
\quad\quad\quad \texttt{st(cntr)}\ = \ulcorner eval(\mathcal{E}^{src}) \urcorner \\
\quad\quad\quad \wedge \\
\quad\quad\quad \texttt{st(cntr -1)}\ = \texttt{st(cntr -2)}\ , \\
\quad\quad\quad \psi^{bc}_{exc}) \\
, \\
\quad\quad \psi^{bc}_{exc}) \\
\quad \wedge \\
\quad \texttt{st(cntr)}\ = ref_{Class}, \\
\quad \psi^{bc}_{exc})
\end{array}
$$

$\{$ *applying twice the lemma* **??** *on page* **??** $\}$

$$
\begin{array}{l}
= \\
wp^{bc}(\ \texttt{new}\ \ Class\ , \\
\quad wp^{bc}(\ \texttt{dup}\ ;\ , \\
\quad\quad wp^{bc}(\ \ulcorner \mathcal{E}^{src} \urcorner\ , \\
\quad\quad\quad \ulcorner \psi^{pre}(ConsClass) \urcorner^{spec} \left\{ \begin{array}{l} [\ulcorner \mathbf{this} \urcorner^{spec} \leftarrow \texttt{st(cntr - 1)}\ ] \\ [\ulcorner arg_1 \urcorner^{spec} \leftarrow \texttt{st(cntr)}\ ] \end{array} \right. \\
\quad\quad\quad \wedge \\
\quad\quad\quad \forall\ mod_i.i = 1..n \\
\quad\quad\quad \left\{ \begin{array}{l}
\ulcorner \mathsf{nPost}^{src}(ConsClass) \urcorner^{spec} \left\{ \begin{array}{l} [\ulcorner \mathbf{this} \urcorner^{spec} \leftarrow \texttt{st(cntr - 1)}\ ] \\ [\ulcorner arg_1 \urcorner^{spec} \leftarrow \texttt{st(cntr)}\ ] \end{array} \right. \Rightarrow \mathsf{nPost}^{src} \\
\wedge \\
exc^{bc}_{ConsClass}(\texttt{Exc}_\texttt{1}) \Rightarrow \mathsf{ePost}^{\mathsf{src}}(\texttt{Exc}_\texttt{1}, \texttt{newClass}(\mathcal{E}^{\mathsf{src}})) \\
\wedge \\
\ldots \\
\wedge \\
exc^{bc}_{ConsClass}(\texttt{Exc}_\texttt{s}) \Rightarrow \mathsf{ePost}^{\mathsf{src}}(\texttt{Exc}_\texttt{s}, \texttt{newClass}(\mathcal{E}^{\mathsf{src}}))
\end{array} \right. \\
\quad\quad\quad \wedge \\
\quad\quad\quad \texttt{st(cntr)}\ = \ulcorner eval(\mathcal{E}^{src}) \urcorner \\
\quad\quad\quad \wedge \\
\quad\quad\quad \texttt{st(cntr -1)}\ = \texttt{st(cntr -2)} \\
\quad\quad\quad \wedge \\
\quad\quad\quad \texttt{st(cntr - 2)}\ = ref_{Class}, \\
\quad\quad\quad \psi^{bc}_{exc}) \\
, \\
\quad\quad \psi^{bc}_{exc}), \\
\quad \psi^{bc}_{exc})
\end{array}
$$

(2)

$=$

$$
\begin{array}{l}
wp^{bc}(\ \texttt{new}\ \ Class\ ,\\
\quad wp^{bc}(\ \texttt{dup}\ ;\ ,\\
\quad\quad wp^{bc}(\ \ulcorner \mathcal{E}^{src} \urcorner\ ,\\
\quad\quad\quad \ulcorner \psi^{pre}(ConsClass) \urcorner^{spec} \left\{ \begin{array}{l} [\ulcorner \mathbf{this} \urcorner^{spec} \leftarrow ref_{Class}] \\ [\ulcorner arg_1 \urcorner^{spec} \leftarrow \ulcorner eval(\mathcal{E}^{src}) \urcorner] \end{array} \right. \\
\quad\quad\quad \wedge \\
\quad\quad\quad \forall\ mod_i.i = 1..n \\
\quad\quad\quad \left\{ \begin{array}{l}
\ulcorner \mathsf{nPost}^{src}(ConsClass) \urcorner^{spec} \left\{ \begin{array}{l} [\ulcorner \mathbf{this} \urcorner^{spec} \leftarrow ref_{Class}] \\ [\ulcorner arg_1 \urcorner^{spec} \leftarrow \ulcorner eval(\mathcal{E}^{src}) \urcorner] \end{array} \right. \quad \Rightarrow \mathsf{nPost}^{src} \\
\wedge \\
exc^{bc}_{ConsClass}(\texttt{Exc}_\texttt{1}) \Rightarrow \mathsf{ePost}^{\mathsf{src}}(\texttt{Exc}_\texttt{1}, \texttt{newClass}(\mathcal{E}^{\mathsf{src}})) \\
\wedge \\
\ldots \\
\wedge \\
exc^{bc}_{ConsClass}(\texttt{Exc}_\texttt{s}) \Rightarrow \mathsf{ePost}^{\mathsf{src}}(\texttt{Exc}_\texttt{s}, \texttt{newClass}(\mathcal{E}^{\mathsf{src}}))
\end{array} \right. \\
, \\
\quad\quad\quad \psi^{bc}_{exc}) \\
, \\
\quad\quad \psi^{bc}_{exc}), \\
\quad \psi^{bc}_{exc})
\end{array}
$$

{ *applying the induction hypothesis we can conclude that the proposition holds for this case* }

The previous lemme gives us the relation between the precondition of a source statement $\mathcal{STMT}$ and its compilation $\ulcorner \mathcal{STMT} \urcorner$ given that the precondition does not contain any stack expressions. Still, our initial definition of $\mathrm{wp}^{\mathrm{bc}}$ (see the discussion in Section **??**) works with implicite postconditions (the function *inter* defined in [3]) which depend on the instructions executed before. In order to establish that the initial definition that we gave of the weakest predicate transformer preserves the proof obligations w.r.t. to the source language we have to establish first the following property:

remind the definition of *inter*

Lemme 1 *For any method* $\texttt{m}$ *with precondition Pre and postcondition Post, for any statement* $\mathcal{STMT}$ *in* $\texttt{body}(\texttt{m})$ *the function* $inter(last\ulcorner \mathcal{STMT} \urcorner, next(last\ulcorner \mathcal{STMT} \urcorner)) \in \mathcal{F}^{bc}_{no\ stack}$

The proof is by structural induction over the structure of a source statement. This lemme means that the postconditions determined by the function *inter* for the compilation $\ulcorner \mathcal{E}^{src} \urcorner$ of an expression $\mathcal{E}^{src}$ is in $\mathcal{F}^{bc}_{no\ stack}$. From property 2 on page 11 we conclude that for any statement $\mathcal{E}$ there exists a formula $\psi \in \mathcal{F}^{src}$ such that $inter(last\ulcorner \mathcal{STMT} \urcorner, \mathrm{next}(last\ulcorner \mathcal{STMT} \urcorner)) = \ulcorner \psi \urcorner$.

Thus, we can conclude that:

$$
\begin{array}{l}
\forall \mathcal{STMT}.\exists\ \psi\ \in \mathcal{F}^{src} \\
inter(last\ulcorner \mathcal{STMT} \urcorner, \mathrm{next}(last\ulcorner \mathcal{STMT} \urcorner)) = \ulcorner \psi \urcorner \\
\wedge \\
\quad \mathrm{wp}^{src}(\ \mathcal{STMT}\ , \psi, \psi^{bc}_{exc}) \\
\quad =_{mod\ Names\ and\ bools} \\
\quad \mathrm{wp}^{bc}(\ \ulcorner \mathcal{STMT} \urcorner\ , inter(last\ulcorner \mathcal{STMT} \urcorner, \mathrm{next}(last\ulcorner \mathcal{STMT} \urcorner)), \psi^{bc}_{exc})
\end{array}
$$

This establishes the equivalence of the preconditions modulo names over source and bytecode programs.

References

[1] Lilian Burdy and Mariela Pavlova. From JML to BCSL. Technical report, INRIA, Sophia-Antipolis, 2004. Draft version. Available from `http://www.inria.fr/everest/Mariela.Pavlova`.

[2] Tim Lindholm and Frank Yellin. Java virtual machine specification. Technical report, Java Software, Sun Microsystems, Inc., 2004.

[3] Mariela Pavlova. Bytecode specification and verification. Technical report, INRIA, Sophia-Antipolis, 2005. Draft version. Available from `http://www.inria.fr/everest/Mariela.Pavlova`.