Preservation of Proof Obligations: PPO

Benjamin Grégoire

INRIA Sophia Antipolis

Types Summer School
August 30th

Plan

- Source language: WHILE
- Bytecode language: JVMI
- Compilation Scheme (correctness)
- A simple VCgen for WHILE
- A simple VCgen for JVMI (soundness)
- Preservation of proof obligations: PPO

Syntax of the source language: WHILE

$$
\begin{array}{llcll}
\text{operations} & \mathrm{op} & ::= & + \mid \times \mid \ldots & \\
\text{comparisons} & \mathrm{cmp} & ::= & \leq \mid = \mid \ldots & \\
\text{expressions} & e & ::= & x \mid c \mid e \ \mathrm{op} \ e & \\
\text{tests} & t & ::= & e \ \mathrm{cmp} \ e & \\
\text{instructions} & i & ::= & x := e & \text{assignment} \\
 & & \mid & \text{if}(t)\{i\}\{i\} & \text{conditional} \\
 & & \mid & \text{while}(t)\{i\} & \text{loop} \\
 & & \mid & i; i & \text{sequence} \\
 & & \mid & \text{skip} & \text{skip}
\end{array}
$$

where $c \in \mathbb{Z}$ and $x \in \mathcal{X}$.

A WHILE program $\mathcal{P} = i; \text{return } e$

Semantics of WHILE

Semantics of expressions $e \overset{\rho}{\hookrightarrow} v$:

$$\frac{}{x \overset{\rho}{\hookrightarrow} \rho(x)} \qquad \frac{}{c \overset{\rho}{\hookrightarrow} c} \qquad \frac{e_1 \overset{\rho}{\hookrightarrow} v_1 \quad e_2 \overset{\rho}{\hookrightarrow} v_2}{e_1 \ \mathrm{op}\ e_2 \overset{\rho}{\hookrightarrow} v_1 \ \mathrm{op}\ v_2}$$

Semantics of instructions $[i,\ \rho] \Downarrow_{\mathcal{S}} \rho'$:

$$\frac{}{[\mathsf{skip},\ \rho] \Downarrow_{\mathcal{S}} \rho}$$

$$\frac{e \overset{\rho}{\hookrightarrow} v}{[x := e,\ \rho] \Downarrow_{\mathcal{S}} \rho\{x \mapsto v\}}$$

$$\frac{[i_1,\ \rho] \Downarrow_{\mathcal{S}} \rho' \quad [\rho',\ i_2] \Downarrow_{\mathcal{S}} \rho''}{[i_1; i_2,\ \rho] \Downarrow_{\mathcal{S}} \rho''}$$

Semantics of branching instructions

$$\frac{e_1 \overset{\rho}{\hookrightarrow} v_1 \quad e_2 \overset{\rho}{\hookrightarrow} v_2}{e_1 \ \mathrm{cmp} \ e_2 \overset{\rho}{\hookrightarrow} v_1 \ \mathrm{cmp} \ v_2}$$

$$\frac{t \overset{\rho}{\hookrightarrow} \mathsf{true} \quad [i_t, \rho] \Downarrow_{\mathcal{S}} \rho'}{[\mathsf{if}(t)\{i_t\}\{i_f\}, \rho] \Downarrow_{\mathcal{S}} \rho'} \qquad \frac{t \overset{\rho}{\hookrightarrow} \mathsf{false} \quad [i_f, \rho] \Downarrow_{\mathcal{S}} \rho'}{[\mathsf{if}(t)\{i_t\}\{i_f\}, \rho] \Downarrow_{\mathcal{S}} \rho'}$$

$$\frac{t \overset{\rho}{\hookrightarrow} \mathsf{false}}{[\mathsf{while}(t)\{i\}, \rho] \Downarrow_{\mathcal{S}} \rho}$$

$$\frac{t \overset{\rho}{\hookrightarrow} \mathsf{true} \quad [i, \rho] \Downarrow_{\mathcal{S}} \rho' \quad [\mathsf{while}(t)\{i\}, \rho'] \Downarrow_{\mathcal{S}} \rho''}{[\mathsf{while}(t)\{i\}, \rho] \Downarrow_{\mathcal{S}} \rho''}$$

Semantics of programs $\mathcal{P} : \rho_0 \Downarrow_{\mathcal{S}} v$

$$\frac{\mathcal{P} = i; \text{return } e \qquad [i,\ \rho_0] \Downarrow_{\mathcal{S}} \rho \qquad e \overset{\rho}{\hookrightarrow} v}{\mathcal{P} : \rho_0 \Downarrow_{\mathcal{S}} v}$$

Remark: We can only express the semantics of terminating programs, to express the semantics of all programs use a small-step semantics (do it !!!).

The JVMI: Informal definitions

A machine state =

bytecode, program counter, operand stack, memory

Bytecode = an array of basic instructions (no more structure)

Program counter, label = a position in the bytecode

Operand stack = a stack used to store intermediate values

(Local) memory = valuation of variables (same as for WHILE)

Syntax of JVMI

intructions	i	$::=$	Iconst c	push value on top of stack
		$\mid$	Ibinop op	binary operation on stack
		$\mid$	Iload x	load value of x on stack
		$\mid$	Istore x	store top of stack in variable x
		$\mid$	Igoto j	unconditional jump
		$\mid$	Iif cmp j	conditional jump
		$\mid$	Ireturn	return the top value of the stack

where $c \in \mathbb{Z}$, $x \in \mathcal{X}$, and $j \in \mathcal{P}_c$.

Semantics of JVMI

$$\frac{\dot{\mathcal{P}}[k] = \mathsf{Iconst}\ c}{\langle k,\ \rho,\ os\rangle \rightsquigarrow \langle k+1,\ \rho,\ c :: os\rangle}$$

$$\frac{\dot{\mathcal{P}}[k] = \mathsf{Ibinop}\ \mathrm{op} \quad v = v_1\ \mathrm{op}\ v_2}{\langle k,\ \rho,\ v_1 :: v_2 :: os\rangle \rightsquigarrow \langle k+1,\ \rho,\ v :: os\rangle}$$

$$\frac{\dot{\mathcal{P}}[k] = \mathsf{Iload}\ x}{\langle k,\ \rho,\ os\rangle \rightsquigarrow \langle k+1,\ \rho,\ \rho(x) :: os\rangle}$$

$$\frac{\dot{\mathcal{P}}[k] = \mathsf{Istore}\ x}{\langle k,\ \rho,\ v :: os\rangle \rightsquigarrow \langle k+1,\ \rho\{x \mapsto v\},\ os\rangle}$$

$$\frac{\dot{\mathcal{P}}[k] = \mathsf{Igoto}\ j}{\langle k,\ \rho,\ os\rangle \rightsquigarrow \langle j,\ \rho,\ os\rangle}$$

$$\frac{\dot{\mathcal{P}}[k] = \mathsf{lif}\ \mathrm{cmp}\ j \quad v_1\ \mathrm{cmp}\ v_2 = \mathsf{true}}{\langle k,\ \rho,\ v_1 :: v_2 :: os\rangle \rightsquigarrow \langle k+1,\ \rho,\ os\rangle}$$

$$\frac{\dot{\mathcal{P}}[k] = \mathsf{lif}\ \mathrm{cmp}\ j \quad v_1\ \mathrm{cmp}\ v_2 = \mathsf{false}}{\langle k,\ \rho,\ v_1 :: v_2 :: os\rangle \rightsquigarrow \langle j,\ \rho,\ os\rangle}$$

Semantics of a bytecode program

$$\frac{\langle 1,\ \rho_0,\ \emptyset\rangle \leadsto^* \langle k,\ \rho,\ v :: os\rangle \qquad \dot{\mathcal{P}}[k] = \mathsf{Ireturn}}{\dot{\mathcal{P}} : \rho_0 \Downarrow v}$$

Compilation scheme

The compiler is defined by two functions:

- Compilation of expressions $[\![e]\!]$:
  generates a bytecode sequence which evaluate e and store/push the result on the top of the operand stack;
- Compilation of instructions $k : [\![i]\!]$:
  k indicates the starting position of the resulting bytecode sequence. It is used to compute the labels attached to branching instructions.

Compilation of expressions

$$
\begin{array}{rcl}
[\![x]\!] & = & \text{Iload } x \\
\langle k,\ \rho,\ os\rangle & \rightsquigarrow & \langle k+1,\ \rho,\ \rho(x) :: os\rangle \\
\\
[\![c]\!] & = & \text{Iconst } c \\
\langle k,\ \rho,\ os\rangle & \rightsquigarrow & \langle k+1,\ \rho,\ c :: os\rangle \\
\\
[\![e_1\ op\ e_2]\!] & = & [\![e_2]\!];\ [\![e_1]\!];\ \text{Ibinop } op \\
\langle k,\ \rho,\ v_1 :: v_2 :: os\rangle & \rightsquigarrow & \langle k+1,\ \rho,\ v_1\ op\ v_2 :: os\rangle
\end{array}
$$

Compilation of instructions

$$
\begin{array}{rcl}
k : [\![x := e]\!] & = & [\![e]\!];\ \text{Istore}\ x \\
\\
k : [\![i_1 ; i_2]\!] & = & k : [\![i_1]\!]; k_2 : [\![i_2]\!] \\
\text{where } k_2 & = & k + |[\![i_1]\!]| \\
\\
k : [\![\text{return}\ e]\!] & = & [\![e]\!];\ \text{Ireturn}
\end{array}
$$

Compilation of instructions

$$
\begin{array}{rcl}
k : [\![\mathrm{if}(e_1 \; cmp \; e_2)\{i_1\}\{i_2\}]\!] & = & [\![e_2]\!];\; [\![e_1]\!];\; \mathrm{Iif}\; cmp\; k_2; \\
 & & k_1 : [\![i_1]\!];\; \mathrm{Igoto}\; k_3;\; k_2 : [\![i_2]\!] \\
\text{where } k_1 & = & k + |[\![e_2]\!]| + |[\![e_1]\!]| + 1 \\
k_2 & = & k_1 + |[\![i_1]\!]| + 1 \\
k_3 & = & k_2 + |[\![i_2]\!]|
\end{array}
$$

$$
\begin{array}{rcl}
k : [\![\mathrm{while}(e_1 \; cmp \; e_2)\{i\}]\!] & = & [\![e_2]\!];\; [\![e_1]\!];\; \mathrm{Iif}\; cmp\; k_2; \\
 & & k_1 : [\![i]\!];\; \mathrm{Igoto}\; k \\
\text{where } k_1 & = & k + |[\![e_2]\!]| + |[\![e_1]\!]| + 1 \\
k_2 & = & k_1 + |[\![i]\!]| + 1
\end{array}
$$

Correctness of the compiler

Lemma (Correctness for expressions)

For all bytecode program $\dot{\mathcal{P}}$, expression e, value v, memory ρ and operand stack os such that $l = |[\![e]\!]|$ and $\dot{\mathcal{P}}[k..k+l] = [\![e]\!]$

$$e \overset{\rho}{\hookrightarrow} v \Rightarrow \langle k,\ \rho,\ os\rangle \rightsquigarrow^* \langle k+l,\ \rho,\ v :: os\rangle$$

Lemma (Correctness for instructions)

For all bytecode program $\dot{\mathcal{P}}$, instruction i, memories ρ and ρ' such that $l = |[\![i]\!]|$ and $\dot{\mathcal{P}}[k..k+l] = k : [\![i]\!]$

$$[i,\ \rho] \Downarrow_{\mathcal{S}} \rho' \Rightarrow \langle k,\ \rho,\ \emptyset\rangle \rightsquigarrow^* \langle k+l,\ \rho',\ \emptyset\rangle$$

Correctness of the compiler

Lemma (Correctness of the compiler)

For all source program $\mathcal{P}$, if $\mathcal{P} : \rho_0 \Downarrow_{\mathcal{S}} v$ then is compiled version evaluate to the same result:

$$\mathcal{P} : \rho_0 \Downarrow_{\mathcal{S}} v \Rightarrow [\![\mathcal{P}]\!] : \rho_0 \Downarrow v$$

Definition (Hoare triple: $\{P\}\ i\ \{Q\}$)

If the value associated to the variables before the execution of the instruction i satisfy the proposition P (precondition) then the value associated to the variables after the execution of i satisfy the proposition Q (postcondition).

Example of rules:

$$\frac{}{\{P\{x \mapsto e\}\}\ x := e\ \{P\}} \qquad \frac{P_1 \Rightarrow P_2 \quad \{P_2\}\ i\ \{Q\}}{\{P_2\}\ i\ \{Q\}}$$

Definition (assertion)

The set of propositions is defined as follow:

$$
\begin{array}{llcl}
\text{Expressions} & \bar{e}(V) & ::= & V \mid c \mid \bar{e}\ op\ \bar{e} \\
\text{Propositions} & P(V) & ::= & \bar{e}(V)\ cmp\ \bar{e}(V) \mid \neg P(V) \\
 & & \mid & P(V) \wedge P(V) \mid P(V) \Rightarrow P(V) \\
\text{Preconditions} & \Phi & ::= & P(\bar{x}) \\
\text{Assertions} & \phi, \psi & ::= & P(x|\bar{x}) \\
\text{Postconditions} & \Psi & ::= & P(\bar{x}|\text{res})
\end{array}
$$

where $\bar{x}$ is a special variable representing the initial value of the variable x, and res is a special value representing the final value of the evaluation of the program.

Definition

Interpretation

- Interpretation of precondition
  $\bar{\rho} \models \Phi \stackrel{\text{def}}{\equiv} \vdash \Phi\{\bar{x} \mapsto \bar{\rho}(x)\}$
- Interpretation of assertion
  $\bar{\rho}, \rho \models \psi \stackrel{\text{def}}{\equiv} \vdash \psi\{\bar{x} \mapsto \bar{\rho}(x)\}\{x \mapsto \rho(x)\}$
- Interpretation of postcondition
  $\bar{\rho}, v \models \Psi \stackrel{\text{def}}{\equiv} \vdash \psi\{\bar{x} \mapsto \bar{\rho}(x)\}\{\text{res} \mapsto v\}$

Verification Condition generator for WHILE

Given a (annotated) program $\mathcal{P}$ a precondition Φ and a postcondition Ψ we want to find a set of verification conditions $\text{VCgen}_{\mathcal{S}}(\mathcal{P}, \Phi, \mathit{Fpost})$ such that if all the verification conditions are provable we have :

$$\left.\begin{array}{r} \bar{\rho} \models \Phi \\ \mathcal{P} : \bar{\rho} \Downarrow_{\mathcal{S}} v \end{array}\right\} \Rightarrow \bar{\rho}, v \models \Psi$$

Verification Condition generator for WHILE

$$\frac{}{\mathrm{wp}_{\mathcal{S}}(\mathrm{skip}, \psi) = \psi, \emptyset} \qquad \frac{}{\mathrm{wp}_{\mathcal{S}}(x := e, \psi) = \psi\{x \mapsto e\}, \emptyset}$$

$$\frac{\mathrm{wp}_{\mathcal{S}}(i_2, \psi) = \phi_2, \theta_2 \quad \mathrm{wp}_{\mathcal{S}}(i_1, \phi_2) = \phi_1, \theta_1}{\mathrm{wp}_{\mathcal{S}}(i_1; i_2, \psi) = \phi_1, \theta_1 \cup \theta_2}$$

$$\frac{\mathrm{wp}_{\mathcal{S}}(i_t, \psi) = \phi_t, \theta_t \quad \mathrm{wp}_{\mathcal{S}}(i_f, \psi) = \phi_f, \theta_f}{\mathrm{wp}_{\mathcal{S}}(\mathrm{if}(t)\{i_t\}\{i_f\}, \psi) = (t \Rightarrow \phi_t) \wedge (\neg t \Rightarrow \phi_f), \theta_t \cup \theta_f}$$

$$\frac{\mathcal{P} = i; \mathrm{return}\ e \quad \mathrm{wp}_{\mathcal{S}}(i, \Psi\{\mathrm{res} \mapsto e\}) = \phi, \theta}{\mathrm{VCgen}_{\mathcal{S}}(\mathcal{P}, \Phi, \Psi) = \{\Phi \Rightarrow \phi\{\vec{x} \mapsto \vec{\vec{x}}\}\} \cup \theta}$$

Verification condition of loop

Rule for loop:

$$\frac{\{I \wedge t\}\ i\ \{I\}}{\{I\}\ \mathsf{while}(t)\{i\}\ \{I \wedge \neg t\}}$$

Application:

$$\frac{\dfrac{\dfrac{(I \wedge t) \Rightarrow \phi \quad \{\phi\}\ i\ \{I\}}{\{I \wedge t\}\ i\ \{I\}}}{\{I\}\ \mathsf{while}(t)\{i\}\ \{I \wedge \neg t\}} \quad (I \wedge \neg t) \Rightarrow \psi}{\{I\}\ \mathsf{while}(t)\{i\}\ \{\psi\}}$$

Verification condition:

$$\frac{\mathsf{wp}_{\mathcal{S}}(i, I) = \phi, \theta}{\mathsf{wp}_{\mathcal{S}}(\mathsf{while}_I(t)\{i\}, \psi) = I, \{I \Rightarrow (t \Rightarrow \phi) \wedge (\neg t \Rightarrow \psi)\} \cup \theta}$$

Correctness of the VCgen

Lemma (Correctness)

For all program $\mathcal{P}$ *if* $\mathrm{VCgen}_{\mathcal{S}}(\mathcal{P}, \Phi, \Psi)$ *are provable then*

$$\left.\begin{array}{r} \bar{\rho} \models \Phi \\ \mathcal{P} : \bar{\rho} \Downarrow_{\mathcal{S}} v \end{array}\right\} \Rightarrow \bar{\rho}, v \models \Psi$$

A VCgen for bytecode

First difference with WHILE: the assertions should refer to position in the stack

Definition (Bytecode proposition)

Stack expressions	$\bar{os}$	$::=$	$\text{os} \mid \bar{e}(sv) :: \bar{os} \mid \uparrow^k \bar{os}$
Bytecode variables	sv	$::=$	$x \mid \bar{x} \mid \bar{os}[i]$
Preconditions	Φ	$::=$	$P(\bar{x})$
Assertions	ϕ, ψ	$::=$	$P(sv)$
Postconditions	Ψ	$::=$	$P(\bar{x} \mid \text{res})$

Second difference with $\text{W}\small{\text{HILE}}$: the loop invariants

Definition

- An annotated bytecode program is a tuple $(\dot{\mathcal{P}}, \Phi, \Lambda, \Psi)$ where Λ is an annotation table.
- An annotation table associate to some program points an assertion (invariant) which should be valid each time the evaluation of the program reach the corresponding program point

Rules of the VCgen

The verification condition generator is defined with two mutually recursive functions $\text{wp}_l(k)$ and $\text{wp}_i(k)$

- $\text{wp}_l(k)$ compute the weakest precondition of the program point k using the annotation table:

$$\text{wp}_l(k) = \begin{cases} \phi \text{ if } \Lambda(k) = \phi \\ \text{wp}_i(k) \end{cases}$$

- $\text{wp}_i(k)$ is the predicate transformer, first the function compute the weakest precondition of all the successors of the instruction at k and then transform the resulting conditions depending on the instruction

Weakest precondition

$$
\begin{array}{lrcl}
 & \dot{\mathcal{P}}[k] & & \\
\text{Iconst } c & \mathrm{wp}_i(k) & = & \mathrm{wp}_l(k+1)\{\mathrm{os} \mapsto c :: \mathrm{os}\} \\
\text{Ibinop } op & \mathrm{wp}_i(k) & = & \mathrm{wp}_l(k+1)\{\mathrm{os} \mapsto (\mathrm{os}[0]\ op\ \mathrm{os}[1]) ::\uparrow^2 \mathrm{os}\} \\
\text{Iload } x & \mathrm{wp}_i(k) & = & \mathrm{wp}_l(k+1)\{\mathrm{os} \mapsto x :: \mathrm{os}\} \\
\text{Istore } x & \mathrm{wp}_i(k) & = & \mathrm{wp}_l(k+1)\{\mathrm{os}, x \mapsto \uparrow \mathrm{os}, \mathrm{os}[0]\} \\
\text{Igoto } l & \mathrm{wp}_i(k) & = & \mathrm{wp}_l(l)
\end{array}
$$

$$
\begin{array}{lrcl}
\text{Iif } cmp\ l & \mathrm{wp}_i(k) & = & \begin{array}{ll} & (t \Rightarrow \mathrm{wp}_l(k+1)\{\mathrm{os} \mapsto \uparrow^2 \mathrm{os}\}) \\ \wedge & (\neg t \Rightarrow \mathrm{wp}_l(l)\{\mathrm{os} \mapsto \uparrow^2 \mathrm{os}\}) \end{array} \\
 & \multicolumn{3}{l}{\text{where } t = \mathrm{os}[0]\ cmp\ \mathrm{os}[1]}
\end{array}
$$

$$
\begin{array}{lrcl}
\text{Ireturn} & \mathrm{wp}_i(k) & = & \Psi\{\mathrm{res} \mapsto \mathrm{os}[0]\}
\end{array}
$$

Definition (VCgen for JVMI)

The set of verification condition of a bytecode program $\mathrm{VCgen}_{\mathcal{B}}(\dot{\mathcal{P}}, \Phi, \Lambda, \Psi)$ is the the smallest set of propositions such that:

- The precondition implies the weakest precondition of the starting point is in the set:

$$(\Phi \Rightarrow \mathrm{wp}_I(0)\{\vec{x} \mapsto \bar{\vec{x}}\}) \in \mathrm{VCgen}_{\mathcal{B}}(\dot{\mathcal{P}}, \Phi, \Lambda, \Psi)$$

- For all annotated program point $(\Lambda(k) = \dot{P})$, the annotation $\dot{P}$ implies the weakest precondition of the instruction at k are in the set:

$$\forall k, \Lambda(k) = \dot{P} \Rightarrow (\dot{P} \Rightarrow \mathrm{wp}_i(k)) \in \mathrm{VCgen}_{\mathcal{B}}(\dot{\mathcal{P}}, \Phi, \Lambda, \Psi)$$

Correctness of the VCgen

Lemma

For all bytecode program $\dot{\mathcal{P}}$, precondition Φ, postcondition, Ψ and annotation table Λ, if the proof obligations of $\dot{\mathcal{P}}$ are valid (i.e. $\vdash \mathrm{VCgen}_{\mathcal{B}}(\dot{\mathcal{P}}, \Phi, \Lambda, \Psi)$) then the following property hold:

$$\bar{\rho}, \rho, os \models \mathrm{wp}_l(k) \Rightarrow \bar{\rho}, \rho, os \models \mathrm{wp}_i(k)$$

Lemma (Soundness for one execution step)

For all bytecode program $\dot{\mathcal{P}}$, precondition Φ, postcondition, Ψ and annotation table Λ, if the proof obligations of $\dot{\mathcal{P}}$ are valid (i.e. $\vdash \mathrm{VCgen}_{\mathcal{B}}(\dot{\mathcal{P}}, \Phi, \Lambda, \Psi)$) then the following property hold:

$$\left.\begin{array}{l} \bar{\rho}, \rho, os \models \mathrm{wp}_i(k) \\ \langle k,\ \rho,\ os \rangle \rightsquigarrow \langle k',\ \rho',\ os' \rangle \end{array}\right\} \Rightarrow \bar{\rho}, \rho', os' \models \mathrm{wp}_l(k')$$

Lemma (Soundness of the bytecode VCgen)

For all bytecode program $\dot{\mathcal{P}}$, precondition Φ, postcondition, Ψ and annotation table Λ, if the proof obligations of $\dot{\mathcal{P}}$ are valid (i.e. $\vdash \mathsf{VCgen}_{\mathcal{B}}(\dot{\mathcal{P}}, \Phi, \Lambda, \Psi)$) then the following property hold:

$$\left.\begin{array}{l}\bar{\rho} \models \Phi \\ \dot{\mathcal{P}} : \bar{\rho} \Downarrow v\end{array}\right\} \Rightarrow \bar{\rho}, v \models \Psi$$

Preservation of proof obligations

Our goal is to show the preservation of proof obligation, i.e. given an annotated source program and his compiled version there exists an annotation table Λ such that:

$$\mathsf{VCgen}_{\mathcal{S}}(\mathcal{P}, \Phi, \Psi) = \mathsf{VCgen}_{\mathcal{B}}([\![\mathcal{P}]\!], \Phi, \Lambda, \Psi)$$

To that end, we extend the compiler for annotated source program. Only the compilation rule for the while change: each time the compiler translate a annotated loop starting from position k ($k : [\![\mathsf{while}_I(t)\{c\}]\!]$), it inserts in the annotation table the invariant I at position k.
The translation of the pre and postcondition is the identity.

Preservation of proof obligations

Lemma (Preservation of proof obligations for expressions)

Given a annotated source program $\mathcal{P}, \Phi, \Psi)$ *and its compiled* $(\dot{\mathcal{P}}, \Phi, \Lambda, \Psi)$. *For all sub-expression e, appearing in the program, if the sequence of code corresponding to the compilation of e start at position k and terminate at position l (i.e.* $l = k + |[\![e]\!]|$*) and* $\mathsf{wp}_l(l) = \psi$ *then* $\mathsf{wp}_l(k) = \psi\{\mathsf{os} \mapsto e :: \mathsf{os}\}$.

Lemma (Preservation of proof obligations for instructions)

Given a annotated program $(i'; \text{return } e', \Phi, \Psi)$ *and its compiled* $(\dot{\mathcal{P}}, \Phi, \Lambda, \Psi)$. *For all sub-instruction* $i \subseteq i'$ *which is compiled starting from position* k *(i.e* $\dot{\mathcal{P}}[k..k + |[\![i]\!]|] = k : [\![i]\!]$*) and for all postcondition* ψ, *if* $\text{wp}_{\mathcal{S}}(i, \psi) = \phi, \theta$ *and* $\text{wp}_{I}(k + |[\![i]\!]|) = \psi$ *then following properties hold:*

- $\text{wp}_I(k) = \phi$
- *For all* $C \in \theta$ *there exists* $k' \in [k..k + |[\![i]\!]|]$ *and loop invariant* I *such that* $\Lambda(k') = I$ *and*

$$C = (I \Rightarrow \text{wp}_i(k'))$$

Soundness of the source VCgen

Soundness of the bytecode VCgen

+

Correctness of the compiler

+

Preservation of proof obligation

==============================

Soundness of the source VCgen