Compositional Natural Semantics and Hoare Logics for Low-Level Languages

Tarmo Uustalu

joint work with Ando Saabas

Estonian CS Theory Days, Viinistu, 28–30 Oct. 2005

MOTIVATION

- Reasoning about low-level code is important in the context of proof-carrying code (PCC), where proofs must be produced for compiled code to avoid the need to trust a compiler.
- This is considered to be unavoidably clumsy as low-level code is believed to be inherently non-modular: in particular, it is believed that there cannot be compositional logics for low-level languages.

This talk

- We present a compositional natural semantics and a matching Hoare logic for a basic low-level language.
- This is based on two ideas:
  - Pieces of low-level code have an inherent partial commutative monoidal structure given by finite unions of pieces of code with non-overlapping supports. Despite its ambiguity, it makes a perfect phrase structure.
  - Differently from statements of a high-level language, pieces of low-level code are multiple-entry and multiple-exit.
- The logic supports compilation of proofs alongside programs.
- Ando: An extension for a stack-based low-level language and on type systems.

OUTLINE

- Syntax, natural semantics and Hoare logic of WHILE, a basic high-level language
- Syntax, natural semantics and Hoare logic of SGOTO, a structured version of a basic low-level language GOTO
- Compilation from WHILE to SGOTO
- Compilation from SGOTO to WHILE

Syntax of While

- There is a supply of program variables $x \in \mathbf{Var}$.
- Arithmetic expressions $a \in \mathbf{AExp}$, boolean expressions $b \in \mathbf{BExp}$ and statements $s \in \mathbf{Stm}$ are defined by the grammar

$$
\begin{array}{lcl}
a & ::= & x \mid n \mid a_0 + a_1 \mid \ldots \\
b & ::= & a_0 = a_1 \mid \ldots \mid \mathrm{tt} \mid \mathrm{ff} \mid \neg b \mid \ldots \\
s & ::= & x := a \mid \mathsf{skip} \mid s_0; s_1 \mid \mathsf{if}\ b\ \mathsf{then}\ s_0\ \mathsf{else}\ s_1 \mid \mathsf{while}\ b\ \mathsf{do}\ s
\end{array}
$$

Natural semantics of While

- States $\sigma \in \textbf{State}$ are stores, i.e., mappings of program variables to integers: $\textbf{State} =_{\mathrm{df}} \textbf{Store} =_{\mathrm{df}} \textbf{Var} \to \mathbb{Z}$.
- Natural semantics rules:

$$\frac{}{\sigma \succ x := a \to \sigma[x \mapsto [\![a]\!]\sigma]} \ :=_{\mathrm{ns}}$$

$$\frac{}{\sigma \succ \mathsf{skip} \to \sigma} \ \mathrm{skip}_{\mathrm{ns}} \qquad \frac{\sigma \succ s_0 \to \sigma'' \quad \sigma'' \succ s_1 \to \sigma'}{\sigma \succ s_0; s_1 \to \sigma'} \ \mathrm{comp}_{\mathrm{ns}}$$

$$\frac{\sigma \models b \quad \sigma \succ s_t \to \sigma'}{\sigma \succ \mathsf{if}\ b\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f \to \sigma'} \ \mathrm{if}^{\mathrm{tt}}_{\mathrm{ns}} \qquad \frac{\sigma \not\models b \quad \sigma \succ s_f \to \sigma'}{\sigma \succ \mathsf{if}\ b\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f \to \sigma'} \ \mathrm{if}^{\mathrm{ff}}_{\mathrm{ns}}$$

$$\frac{\sigma \models b \quad \sigma \succ s \to \sigma'' \quad \sigma'' \succ \mathsf{while}\ b\ \mathsf{do}\ s \to \sigma'}{\sigma \succ \mathsf{while}\ b\ \mathsf{do}\ s \to \sigma'} \ \mathrm{while}^{\mathrm{tt}}_{\mathrm{ns}}$$

$$\frac{\sigma \not\models b}{\sigma \succ \mathsf{while}\ b\ \mathsf{do}\ s \to \sigma} \ \mathrm{while}^{\mathrm{ff}}_{\mathrm{ns}}$$

Hoare logic of While

- Assertions $P \in \mathbf{Assn}$ are logic formulae over the signature of arithmetic extended with the program variables $x \in \mathbf{Var}$ as constants.
- Hoare rules:

$$\frac{}{\{Q[x \mapsto a]\}\, x := a\, \{Q\}} \;:=_{\mathrm{hoa}}$$

$$\frac{}{\{P\}\, \mathsf{skip}\, \{P\}} \;\mathrm{skip}_{\mathrm{hoa}} \qquad \frac{\{P\}\, s_0\, \{R\} \quad \{R\}\, s_1\, \{Q\}}{\{P\}\, s_0; s_1\, \{Q\}} \;\mathrm{comp}_{\mathrm{hoa}}$$

$$\frac{\{b \wedge P\}\, s_t\, \{Q\} \quad \{\neg b \wedge P\}\, s_f\, \{Q\}}{\{P\}\, \mathsf{if}\ b\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f\, \{Q\}} \;\mathrm{if}_{\mathrm{hoa}} \qquad \frac{\{b \wedge P\}\, s\, \{P\}}{\{P\}\, \mathsf{while}\ b\ \mathsf{do}\ s\, \{\neg b \wedge P\}} \;\mathrm{while}_{\mathrm{hoa}}$$

$$\frac{P \models P' \quad \{P'\}\, s\, \{Q'\} \quad Q' \models Q}{\{P\}\, s\, \{Q\}} \;\mathrm{conseq}_{\mathrm{hoa}}$$

- **Theorem (Soundness)** If $\{P\}\, s\, \{Q\}$, then, for any σ, σ' and α, $\sigma \models_\alpha P$ and $\sigma \succ\!\! s \!\!\rightarrow \sigma'$ imply $\sigma' \models_\alpha Q$.

- The weakest liberal precondition of a statement: $\mathrm{wlp}(s, Q)$ is some assertion P such that $\sigma \models_\alpha P$ for a state σ and valuation α iff $\sigma \succ\!\! s \!\!\rightarrow \sigma'$ implies $\sigma' \models_\alpha Q$ for any state σ'.

- **Lemma** $\{\mathrm{wlp}(s, Q)\}\, s\, \{Q\}$.

- **Theorem (Completeness)** If, for any σ, σ' and α, $\sigma \models_\alpha P$ and $\sigma \succ\!\! s \!\!\rightarrow \sigma'$ imply $\sigma' \models_\alpha Q$, then $\{P\}\, s\, \{Q\}$.

Syntax of Goto

- Labels $\ell \in \mathbf{Label}$ are natural numbers: $\mathbf{Label} =_{\mathrm{df}} \mathbb{N}$.
- Instructions $\mathit{instr} \in \mathbf{Instr}$ are defined by the grammar

$$\mathit{instr} \quad ::= \quad x := a \mid \mathsf{goto}\ \ell \mid \mathsf{ifnot}\ b\ \mathsf{goto}\ \ell$$

- Labelled instructions $(\ell, \mathit{instr}) \in \mathbf{LInstr}$ are pairs of labels and instructions: $\mathbf{LInstr} =_{\mathrm{df}} \mathbf{Label} \times \mathbf{Instr}$.
- A piece of code $c \in \mathbf{Code}$ is a finite set of labelled instructions: $\mathbf{Code} = \mathcal{P}_{\mathrm{fin}}(\mathbf{LInstr})$.
- The domain of a piece of code: $\mathrm{dom}(c) =_{\mathrm{df}} \{\ell \mid (\ell, \mathit{instr}) \in c\}$.
- Wellformedness of a piece of code: c is wellformed iff $(\ell, \mathit{instr}) \in c$ and $(\ell, \mathit{instr}') \in c$ imply $\mathit{instr} = \mathit{instr}'$.

SEMANTICS OF GOTO

- States $(\ell, \sigma) \in \mathbf{State}$ are pairs of a label (a value for the pc) and a store (values for the program variables): $\mathbf{State} =_{\mathrm{df}} \mathbf{Label} \times \mathbf{Store}$.
- Single-step reduction is defined by the rules

$$\frac{(\ell, x := a) \in c}{c \vdash (\ell, \sigma) \twoheadrightarrow (\ell + 1, \sigma[x \mapsto [\![a]\!]\sigma])} := \qquad \frac{(\ell, \mathsf{goto}\ m) \in c}{c \vdash (\ell, \sigma) \twoheadrightarrow (m, \sigma)}\ \mathrm{goto}$$

$$\frac{(\ell, \mathsf{ifnot}\ b\ \mathsf{goto}\ m) \in c \quad \sigma \models b}{c \vdash (\ell, \sigma) \twoheadrightarrow (\ell + 1, \sigma)}\ \mathrm{ifngoto}^{\mathrm{tt}} \qquad \frac{(\ell, \mathsf{ifnot}\ b\ \mathsf{goto}\ m) \in c \quad \sigma \not\models b}{c \vdash (\ell, \sigma) \twoheadrightarrow (m, \sigma)}\ \mathrm{ifngoto}^{\mathrm{ff}}$$

  Multi-step reduction is the reflexive-transitive closure.

Syntax of SGoto

- Labels $\ell \in \mathbf{Label}$ are natural numbers: $\mathbf{Label} =_{\mathrm{df}} \mathbb{N}$.
- Instructions $instr \in \mathbf{Instr}$ and pieces of structured code $sc \in \mathbf{SCode}$ are defined by the grammar

$$\begin{array}{rcl} instr & ::= & x := a \mid \mathsf{goto}\ \ell \mid \mathsf{ifnot}\ b\ \mathsf{goto}\ \ell \\ sc & ::= & (\ell, instr) \mid \mathbf{0} \mid sc_0 \oplus sc_1 \end{array}$$

- The domain of a piece of code: $\mathrm{dom}((\ell, instr)) = \{\ell\}$, $\mathrm{dom}(\mathbf{0}) = \emptyset$, and $\mathrm{dom}(sc_0 \oplus sc_1) = \mathrm{dom}(sc_0) \cup \mathrm{dom}(sc_1)$.
- Wellformedness of a piece of code: $(\ell, instr)$ is wellformed, $\mathbf{0}$ is wellformed, and $sc_0 \oplus sc_1$ is wellformed iff both sc_0, and sc_1 are wellformed and $\mathrm{dom}(sc_0) \cap \mathrm{dom}(sc_1) = \emptyset$.
- Forgetful function $U \in \mathbf{SCode} \to \mathbf{Code}$: $U((\ell, instr)) = \{(\ell, instr)\}$, $U(\mathbf{0}) = \emptyset$, $U(sc_0 \oplus sc_1) = U(sc_0) \cup U(sc_1)$.

Natural semantics of SGoto

- States $(\ell, \sigma) \in \mathbf{State}$ are pairs of a pc value and a store $\mathbf{State} =_{\mathrm{df}} \mathbf{Label} \times \mathbf{Store}$.
- Natural semantics rules:

$$\frac{}{(\ell, \sigma) \succ (\ell, x := a) \rightarrow (\ell + 1, \sigma[x \mapsto [\![a]\!]\sigma])} \; :=_{\mathrm{ns}}$$

$$\frac{m \neq \ell}{(\ell, \sigma) \succ (\ell, \mathsf{goto}\ m) \rightarrow (m, \sigma)} \; \mathrm{goto}_{\mathrm{ns}}$$

$$\frac{\sigma \models b}{(\ell, \sigma) \succ (\ell, \mathsf{ifnot}\ b\ \mathsf{goto}\ m) \rightarrow (\ell + 1, \sigma)} \; \mathrm{ifngoto}^{\mathrm{tt}}_{\mathrm{ns}}$$

$$\frac{\sigma \not\models b \quad m \neq \ell}{(\ell, \sigma) \succ (\ell, \mathsf{ifnot}\ b\ \mathsf{goto}\ m) \rightarrow (m, \sigma)} \; \mathrm{ifngoto}^{\mathrm{ff}}_{\mathrm{ns}}$$

$$\frac{\ell \in \mathrm{dom}(sc_0) \quad (\ell,\sigma) \succ sc_0 \rightarrow (\ell',\sigma') \quad (\ell',\sigma') \succ sc_0 \oplus sc_1 \rightarrow (\ell'',\sigma'')}{(\ell,\sigma) \succ sc_0 \oplus sc_1 \rightarrow (\ell'',\sigma'')} \; \oplus^0_{\mathrm{ns}}$$

$$\frac{\ell \in \mathrm{dom}(sc_1) \quad (\ell,\sigma) \succ sc_1 \rightarrow (\ell',\sigma') \quad (\ell',\sigma') \succ sc_0 \oplus sc_1 \rightarrow (\ell'',\sigma'')}{(\ell,\sigma) \succ sc_0 \oplus sc_1 \rightarrow (\ell'',\sigma'')} \; \oplus^1_{\mathrm{ns}}$$

$$\frac{\ell \notin \mathrm{dom}(sc)}{(\ell,\sigma) \succ sc \rightarrow (\ell,\sigma)} \; \mathrm{ood}_{\mathrm{ns}}$$

- **Lemma (Postlabels)** If $(\ell,\sigma) \succ sc \rightarrow (\ell',\sigma')$, then $\ell' \notin \mathrm{dom}(sc)$.
- **Theorem (Preservation of evaluations as stuck reduction sequences)** If $(\ell,\sigma) \succ sc \rightarrow (\ell',\sigma')$, then $U(sc) \vdash (\ell,\sigma) \twoheadrightarrow^* (\ell',\sigma') \not\twoheadrightarrow$.
- **Theorem (Reflection of stuck reduction sequences as evaluations)** If $U(sc) \vdash (\ell_0,\sigma_0) \twoheadrightarrow^k (\ell_k,\sigma_k) \not\twoheadrightarrow$, then $(\ell_0,\sigma_0) \succ sc \rightarrow (\ell_k,\sigma_k)$.

- **Theorem (Neutrality wrt chosen phrase structure)**
  If $U(sc_0) = U(sc_1)$, then $sc_0 \cong sc_1$
  (meaning that $(\ell, \sigma) \succ sc_0 \rightarrow (\ell', \sigma')$ iff $(\ell, \sigma) \succ sc_1 \rightarrow (\ell', \sigma')$ for any states (ℓ, σ), (ℓ', σ')).

- **Corollary (Partial commutative monoidal structure)**

  1. $(sc_0 \oplus sc_1) \oplus sc_2 \cong sc_0 \oplus (sc_1 \oplus sc_2)$,
  2. $\mathbf{0} \oplus sc \cong sc \cong sc \oplus \mathbf{0}$,
  3. $sc_0 \oplus sc_1 \cong sc_1 \oplus sc_0$.

HOARE RULES OF SGOTO

- Assertions are logic formulae over the signature of arithmetic extended with a special symbol pc and program variables $x \in \mathbf{Var}$ as constants.
- Hoare rules:

$$\frac{}{\{(pc = \ell \wedge Q[(pc, x) \mapsto (\ell + 1, a)]) \vee (pc \neq \ell \wedge Q)\}\ (\ell, x := a)\ \{Q\}}\ :=_{\mathrm{hoa}}$$

$$\frac{}{\{(pc = \ell \wedge (Q[pc \mapsto m] \vee m = \ell)) \vee (pc \neq \ell \wedge Q)\}\ (\ell, \mathsf{goto}\ m)\ \{Q\}}\ \mathrm{goto}_{\mathrm{hoa}}$$

$$\frac{}{\left\{\begin{array}{ll} & (pc = \ell\ \wedge\ ((b \wedge Q[pc \mapsto \ell + 1]) \\ & \qquad\qquad \vee\ (\neg b \wedge (Q[pc \mapsto m] \vee m = \ell)))) \\ \vee & (pc \neq \ell\ \wedge\ Q) \end{array}\right\}\ (\ell, \mathsf{ifnot}\ b\ \mathsf{goto}\ m)\ \left\{\ Q\ \right\}}\ \mathrm{ifngoto}_{\mathrm{hoa}}$$

$$\frac{}{\{P\}\,\mathbf{0}\,\{P\}}\;\mathbf{0}_{\mathrm{hoa}}$$

$$\frac{\{pc \in \mathrm{dom}(sc_0) \wedge P\}\, sc_0\, \{P\} \quad \{pc \in \mathrm{dom}(sc_1) \wedge P\}\, sc_1\, \{P\}}{\{P\}\, sc_0 \oplus sc_1\, \{pc \notin \mathrm{dom}(sc_0) \wedge pc \notin \mathrm{dom}(sc_1) \wedge P\}}\;\oplus_{\mathrm{hoa}}$$

$$\frac{P \models P' \quad \{P'\}\, sc\, \{Q'\} \quad Q' \models Q}{\{P\}\, sc\, \{Q\}}\;\mathrm{conseq}_{\mathrm{hoa}}$$

- **Theorem (Soundness)** If $\{P\}\, sc\, \{Q\}$, then, for any ℓ_0, σ_0, ℓ', σ' and α, $(\ell_0, \sigma_0) \models_\alpha P$ and $(\ell_0, \sigma_0) \succ\!sc\!\rightarrow (\ell', \sigma')$ imply $(\ell', \sigma') \models_\alpha Q$.
- The weakest liberal precondition of a piece of code: $\mathrm{wlp}(sc, Q)$ is some assertion P such that $(\ell, \sigma) \models_\alpha P$ for a state (ℓ, σ) and valuation α iff $(\ell, \sigma) \succ\!sc\!\rightarrow (\ell', \sigma')$ implies $(\ell', \sigma') \models_\alpha Q$ for any state (ℓ', σ').
- **Lemma** $\{\mathrm{wlp}(s, Q)\}\, sc\, \{Q\}$.
- **Theorem (Completeness)** If, for any ℓ_0, σ_0, ℓ', σ' and α, $(\ell_0, \sigma_0) \models_\alpha P$ and $(\ell_0, \sigma_0) \succ\!sc\!\rightarrow (\ell', \sigma')$ imply $(\ell', \sigma') \models_\alpha Q$, then $\{P\}\, sc\, \{Q\}$.

Compilation from While to SGoto

- Compilation rules:

$$\frac{}{x := a \;{}^{\ell}\!\searrow_{\ell+1} (\ell, x := a)}$$

$$\frac{}{\mathsf{skip} \;{}^{\ell}\!\searrow_{\ell} \mathbf{0}}$$

$$\frac{s_0 \;{}^{\ell}\!\searrow_{\ell''} sc_0 \qquad s_1 \;{}^{\ell''}\!\searrow_{\ell'} sc_1}{s_0; s_1 \;{}^{\ell}\!\searrow_{\ell'} sc_0 \oplus sc_1}$$

$$\frac{s_t \;{}^{\ell+1}\!\searrow_{\ell''} sc_t \qquad s_f \;{}^{\ell''+1}\!\searrow_{\ell'} sc_f}{\mathsf{if}\ b\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f \;{}^{\ell}\!\searrow_{\ell'} (\ell, \mathsf{ifnot}\ b\ \mathsf{goto}\ \ell' + 1) \oplus ((sc_t \oplus (\ell'', \mathsf{goto}\ \ell')) \oplus sc_f)}$$

$$\frac{s \;{}^{\ell+1}\!\searrow_{\ell''} sc}{\mathsf{while}\ b\ \mathsf{do}\ s \;{}^{\ell}\!\searrow_{\ell''+1} (\ell, \mathsf{ifnot}\ b\ \mathsf{goto}\ \ell'' + 1) \oplus (sc \oplus (\ell'', \mathsf{goto}\ \ell))}$$

- **Lemma (Domain of compiled code)**
  If $s \,{}^{\ell}\!\searrow_{\ell'}\, sc$, then $\mathrm{dom}(sc) = [\ell, \ell')$.

- **Theorem (Preservation of evaluations)**
  If $s \,{}^{\ell}\!\searrow_{\ell'}\, sc$ and $\sigma \succ\!s\!\rightarrow \sigma'$, then $(\ell, \sigma) \succ\!sc\!\rightarrow (\ell', \sigma')$.

- **Theorem (Reflection of evaluations)**
  If $s \,{}^{\ell}\!\searrow_{\ell'}\, sc$ and $(\ell, \sigma) \succ\!sc\!\rightarrow (\ell'', \sigma')$, then $\ell' = \ell''$ and $\sigma \succ\!s\!\rightarrow \sigma'$.

- **Theorem (Preservation of derivable Hoare triples)**
  If $s \,{}^{\ell}\!\searrow_{\ell'}\, sc$ and $\{P\}\, s\, \{Q\}$, then $\{pc = \ell \wedge P\}\, sc\, \{pc = \ell' \wedge Q\}$.

- **Theorem (Reflection of derivable Hoare triples)**
  If $s \,{}^{\ell}\!\searrow_{\ell'}\, sc$ and $\{P\}\, sc\, \{Q\}$, then $\{P[pc \mapsto \ell]\}\, s\, \{Q[pc \mapsto \ell']\}$.

Example

- A WHILE program: $S =_{\mathrm{df}}$ while $x < n$ do $(x := x + 1; s := s * x)$
- A proof:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\left\{\begin{array}{c} x + 1 \leq n \\ \wedge\, s * (x + 1) = (x + 1)! \end{array}\right\}\, x := x + 1\, \left\{\begin{array}{c} x \leq n \\ \wedge\, s * x = x! \end{array}\right\}}}{\{x < n \wedge s = x!\}\, x := x + 1\, \{x \leq n \wedge s * x = x!\}} \quad \dfrac{}{\{x \leq n \wedge s * x = x!\}\, s := s * x\, \{s = x! \wedge x \leq n\}}}{\{x < n \wedge s = x!\}\, x := x + 1; s := s * x\, \{x \leq n \wedge s = x!\}}}{\{x < n \wedge x \leq n \wedge s = x!\}\, x := x + 1; s := s * x\, \{x \leq n \wedge s = x!\}}}{\{x \leq n \wedge s = x!\}\, S\, \{x \not< n \wedge x \leq n \wedge s = x!\}}}{\{x = 0 \wedge s = 1\}\, S\, \{x = n \wedge s = n!\}}
$$

- Compiled program:
  $C =_{\mathrm{df}} (1, \mathsf{ifnot}\ x < n\ \mathsf{goto}\ 5) \oplus (((2, x := x + 1) \oplus (3, s := s * x)) \oplus (4, \mathsf{goto}\ 1))$
- Compiled proof:

$$
\begin{prooftree}
\AxiomC{}
\UnaryInfC{$\{J_{1'}\}\,1\,\{I_{25}\}$}
\UnaryInfC{$\{pc = 1 \wedge I_{1'25}\}\,1\,\{I_{1'25}\}$}
\AxiomC{}
\UnaryInfC{$\{J_{2''}\}\,2\,\{I_3\}$}
\UnaryInfC{$\{I_{2''}\}\,2\,\{I_3\}$}
\UnaryInfC{$\{I_{2'}\}\,2\,\{I_3\}$}
\UnaryInfC{$\{pc = 2 \wedge I_{2'34}\}\,2\,\{I_{2'34}\}$}
\AxiomC{}
\UnaryInfC{$\{J_3\}\,3\,\{I_4\}$}
\UnaryInfC{$\{I_3\}\,3\,\{I_4\}$}
\UnaryInfC{$\{pc = 3 \wedge I_{2'34}\}\,3\,\{I_{2'34}\}$}
\BinaryInfC{$\{I_{2'34}\}\,2 \oplus 3\,\{pc \notin [2,4) \wedge I_{2'34}\}$}
\UnaryInfC{$\{I_2'\}\,2 \oplus 3\,\{I_4\}$}
\UnaryInfC{$\{I_2\}\,2 \oplus 3\,\{I_4\}$}
\UnaryInfC{$\{pc \in [2,4) \wedge I_{1'24}\}\,2 \oplus 3\,\{I_{1'24}\}$}
\AxiomC{}
\UnaryInfC{$\{J_4\}\,4\,\{I_{1'}\}$}
\UnaryInfC{$\{pc = 4 \wedge I_{1'24}\}\,4\,\{I_{1'24}\}$}
\BinaryInfC{$\{I_{1'24}\}\,(2 \oplus 3) \oplus 4\,\{pc \notin [2,5) \wedge I_{1'24}\}$}
\UnaryInfC{$\{pc \in [2,5) \wedge I_{1'25}\}\,(2 \oplus 3) \oplus 4\,\{I_{1'25}\}$}
\BinaryInfC{$\{I_{1'25}\}\,C\,\{pc \notin [1,5) \wedge I_{1'25}\}$}
\UnaryInfC{$\{I_{1'}\}\,C\,\{I_5\}$}
\UnaryInfC{$\{I_1\}\,C\,\{I_{5'}\}$}
\end{prooftree}
$$

where

$$
\begin{aligned}
I_1 &=_{\mathrm{df}} pc = 1 \wedge x = 0 \wedge s = 1 \\
I_{1'} &=_{\mathrm{df}} pc = 1 \wedge x \leq n \wedge s = x! \\
I_2 &=_{\mathrm{df}} pc = 2 \wedge x < n \wedge x \leq n \wedge s = x! \\
I_{2'} &=_{\mathrm{df}} pc = 2 \wedge x < n \wedge s = x! \\
I_{2''} &=_{\mathrm{df}} pc = 2 \wedge x + 1 \leq n \wedge s * (x + 1) = (x + 1)!
\end{aligned}
\qquad
\begin{aligned}
I_3 &=_{\mathrm{df}} pc = 3 \wedge x \leq n \wedge s * x = x! \\
I_4 &=_{\mathrm{df}} pc = 4 \wedge x \leq n \wedge s = x! \\
I_5 &=_{\mathrm{df}} pc = 5 \wedge x \not< n \wedge x \leq n \wedge s = x! \\
I_{5'} &=_{\mathrm{df}} pc = 5 \wedge x = n \wedge s = x!
\end{aligned}
$$

and

$$
\begin{aligned}
J_{1'} &=_{\mathrm{df}} (pc = 1 \wedge ((x < n \wedge I_{25}[pc \mapsto 2]) \vee (x \not< n \wedge (I_{25}[pc \mapsto 5] \vee 5 = 1))) \vee (pc \neq 1 \wedge I_{25}) \\
J_{2''} &=_{\mathrm{df}} (pc = 2 \wedge I_3[(pc, x) \mapsto (2, x + 1)]) \vee (pc \neq 2 \wedge I_3) \\
J_3 &=_{\mathrm{df}} (pc = 3 \wedge I_4[(pc, s) \mapsto (3, s * x)]) \vee (pc \neq 3 \wedge I_4) \\
J_4 &=_{\mathrm{df}} (pc = 4 \wedge (I_{1'}[pc \mapsto 4] \vee 1 = 4)) \vee (pc \neq 4 \wedge I_{1'})
\end{aligned}
$$

Compilation from SGoto to While

- Rules of compilation from SGoto to While:

$$\frac{}{(\ell, x := a) \nearrow \text{if } x_{pc} = \ell \text{ then } x := a; x_{pc} := x_{pc} + 1 \text{ else skip}}$$

$$\frac{}{(\ell, \text{goto } m) \nearrow \text{while } x_{pc} = \ell \text{ do } x_{pc} := m}$$

$$\frac{}{(\ell, \text{ifnot } b \text{ goto } m) \nearrow \text{while } x_{pc} = \ell \text{ do } (\text{if } b \text{ then } x_{pc} := \ell + 1 \text{ else } x_{pc} := m)}$$

$$\frac{}{\mathbf{0} \nearrow \text{skip}}$$

$$\frac{sc_0 \nearrow s_0 \quad sc_1 \nearrow s_1}{sc_0 \oplus sc_1 \nearrow \begin{array}{c} \text{while } x_{pc} \in \mathrm{dom}(sc_0) \vee x_{pc} \in \mathrm{dom}(sc_1) \text{ do} \\ (\text{if } x_{pc} \in \mathrm{dom}(sc_0) \text{ then } s_0 \text{ else } s_1) \end{array}}$$

- **Theorem (Preservation of evaluations)**
  If $sc \nearrow s$ and $(\ell, \sigma) \succ\!\! sc \!\!\rightarrow (\ell', \sigma')$, then $\sigma[x_{pc} \mapsto \ell] \succ\!\! s \!\!\rightarrow \sigma'[x_{pc} \mapsto \ell']$.
- **Theorem (Reflection of evaluations)**
  If $sc \nearrow s$ and $\sigma \succ\!\! s \!\!\rightarrow \sigma'$, then $(\sigma(x_{pc}), \sigma[x_{pc} \mapsto n]) \succ\!\! sc \!\!\rightarrow (\sigma'(x_{pc}), \sigma'[x_{pc} \mapsto n])$.
- **Theorem (Preservation of derivable Hoare triples)**
  If $sc \nearrow s$ and $\{P\}\, sc\, \{Q\}$, then $\{P[pc \mapsto x_{pc}]\}\, s\, \{Q[pc \mapsto x_{pc}]\}$.
- **Theorem (Reflection of derivable Hoare triples)**
  If $sc \nearrow s$ and $\{P\}\, s\, \{Q\}$, then $\{P[x_{pc} \mapsto pc]\}\, sc\, \{Q[x_{pc} \mapsto pc]\}$.

RELATED WORK

- In early days of Hoare logic, considerable attention was paid to structured high-level languages with general or restricted jumps: conditional Hoare triples to make use of label invariants (Clint & Hoare, Kowaltowski, de Bruin), multiple-postcondition Hoare triples to reflect that statements involving gotos are multiple-exit (Arbib & Alagić).
- Reasoning about unstructured low-level language code has attracted interest in relation to PCC. Quigley's work is based on decompilation, Benton's logic makes use of global label invariants as the logic of de Bruin.
- Tan and Appel (2005) use finite unions and the idea that low-level code is multiple-entry, multiple-exit. But their logic is continuation-style and adopts a non-standard interpretation of Hoare triples via approximations of falsity.

CONCLUSION

- Nothing beyond the structure given by finite unions is needed to give a low-level language a compositional natural semantics and Hoare logic with every desirable property.
- The semantic and logic descriptions so obtained are no more complicated than the standard ones for high-level languages.
- The logic description supports compilation of proofs alongside programs.
- The structure of finite unions is natural from practical point of view.