Compositional Type Systems for Stack-Based Low-Level Languages

A. Saabas T. Uustalu

Institute of Cybernetics

Theory Days, Viinistu, 29 October 2005

Background

- In the SOS 2005 paper, we proposed a novel method for developing compositional natural semantics and Hoare logics for languages with jumps

Background

- In the SOS 2005 paper, we proposed a novel method for developing compositional natural semantics and Hoare logics for languages with jumps
- The central idea was to use the structure of finite disjoint unions as a phrase structure.

Background

- In the SOS 2005 paper, we proposed a novel method for developing compositional natural semantics and Hoare logics for languages with jumps
- The central idea was to use the structure of finite disjoint unions as a phrase structure.
- Here, we develop these ideas further, and consider an operand stack based language PUSH

Background

- In the SOS 2005 paper, we proposed a novel method for developing compositional natural semantics and Hoare logics for languages with jumps
- The central idea was to use the structure of finite disjoint unions as a phrase structure.
- Here, we develop these ideas further, and consider an operand stack based language PUSH
  - More demanding since stack errors can occur

Background

- In the SOS 2005 paper, we proposed a novel method for developing compositional natural semantics and Hoare logics for languages with jumps
- The central idea was to use the structure of finite disjoint unions as a phrase structure.
- Here, we develop these ideas further, and consider an operand stack based language PUSH
  - More demanding since stack errors can occur
  - Makes sense to study type systems for attesting code safety

Syntax of PUSH

- The instructions *instr* $\in$ **Instr** are given by the grammar

$$\begin{array}{rcl} instr & ::= & \mathsf{load}\ x \mid \mathsf{store}\ x \mid \mathsf{push}\ n \\ & & \mid \mathsf{add} \mid \mathsf{eq} \mid \ldots \mid \mathsf{goto}\ \ell \mid \mathsf{gotoF}\ \ell \end{array}$$

Syntax of PUSH

- The instructions *instr* $\in$ **Instr** are given by the grammar

$$
\begin{aligned}
\mathit{instr} ::= {} & \mathsf{load}\ x \mid \mathsf{store}\ x \mid \mathsf{push}\ n \\
& \mid \mathsf{add} \mid \mathsf{eq} \mid \ldots \mid \mathsf{goto}\ \ell \mid \mathsf{gotoF}\ \ell
\end{aligned}
$$

- States (ℓ, ζ, σ) are triples of a label (value of the *pc*), stack and store. Stacks are lists of integers and booleans. Store is a mapping from register names to values.

Single-step reduction rules of Push

$$\frac{(\ell, \mathsf{store}\ x) \in c \quad n \in \mathbb{Z}}{c \vdash (\ell, n :: \zeta, \sigma) \twoheadrightarrow (\ell + 1, \zeta, \sigma[x \mapsto n])}\ \text{store}$$

Single-step reduction rules of PUSH

$$\frac{(\ell, \mathsf{store}\ x) \in c \quad n \in \mathbb{Z}}{c \vdash (\ell, n :: \zeta, \sigma) \twoheadrightarrow (\ell + 1, \zeta, \sigma[x \mapsto n])}\ \text{store}$$

$$\frac{(\ell, \mathsf{load}\ x) \in c}{c \vdash (\ell, \zeta, \sigma) \twoheadrightarrow (\ell + 1, \sigma(x) :: \zeta, \sigma)}\ \text{load}$$

Single-step reduction rules of PUSH

$$\frac{(\ell, \mathsf{store}\ x) \in c \quad n \in \mathbb{Z}}{c \vdash (\ell, n :: \zeta, \sigma) \twoheadrightarrow (\ell+1, \zeta, \sigma[x \mapsto n])}\ \text{store}$$

$$\frac{(\ell, \mathsf{load}\ x) \in c}{c \vdash (\ell, \zeta, \sigma) \twoheadrightarrow (\ell+1, \sigma(x) :: \zeta, \sigma)}\ \text{load}$$

$$\ldots$$

$$\frac{(\ell, \mathsf{goto}\ m) \in c}{c \vdash (\ell, \zeta, \sigma) \twoheadrightarrow (m, \zeta, \sigma)}\ \text{goto}$$

Single-step reduction rules of PUSH

$$\frac{(\ell, \mathsf{store}\ x) \in c \quad n \in \mathbb{Z}}{c \vdash (\ell, n :: \zeta, \sigma) \twoheadrightarrow (\ell + 1, \zeta, \sigma[x \mapsto n])}\ \text{store}$$

$$\frac{(\ell, \mathsf{load}\ x) \in c}{c \vdash (\ell, \zeta, \sigma) \twoheadrightarrow (\ell + 1, \sigma(x) :: \zeta, \sigma)}\ \text{load}$$

$$\ldots$$

$$\frac{(\ell, \mathsf{goto}\ m) \in c}{c \vdash (\ell, \zeta, \sigma) \twoheadrightarrow (m, \zeta, \sigma)}\ \text{goto}$$

$$\frac{(\ell, \mathsf{gotoF}\ m) \in c}{c \vdash (\ell, \mathrm{tt} :: \zeta, \sigma) \twoheadrightarrow (\ell + 1, \zeta, \sigma)}\ \text{gotoF}$$

Single-step reduction rules of PUSH

$$\frac{(\ell, \mathsf{store}\ x) \in c \quad n \in \mathbb{Z}}{c \vdash (\ell, n :: \zeta, \sigma) \twoheadrightarrow (\ell + 1, \zeta, \sigma[x \mapsto n])}\ \text{store}$$

$$\frac{(\ell, \mathsf{load}\ x) \in c}{c \vdash (\ell, \zeta, \sigma) \twoheadrightarrow (\ell + 1, \sigma(x) :: \zeta, \sigma)}\ \text{load}$$

$$\ldots$$

$$\frac{(\ell, \mathsf{goto}\ m) \in c}{c \vdash (\ell, \zeta, \sigma) \twoheadrightarrow (m, \zeta, \sigma)}\ \text{goto}$$

$$\frac{(\ell, \mathsf{gotoF}\ m) \in c}{c \vdash (\ell, \mathrm{tt} :: \zeta, \sigma) \twoheadrightarrow (\ell + 1, \zeta, \sigma)}\ \text{gotoF}$$

The associated multi-step reduction relation $\twoheadrightarrow^*$ is the reflexive-transitive closure of the single-step relation.

Structured version of PUSH

- Some structure should be introduced into PUSH code.

Structured version of PUSH

- Some structure should be introduced into PUSH code.
- We use the structure of finite unions of non-overlapping pieces of code.

$$sc ::= (\ell, instr) \mid \mathbf{0} \mid sc_0 \oplus sc_1$$

Structured version of PUSH

- Some structure should be introduced into PUSH code.
- We use the structure of finite unions of non-overlapping pieces of code.

$$sc ::= (\ell, instr) \mid \mathbf{0} \mid sc_0 \oplus sc_1$$

- Domain $\mathrm{dom}(sc)$ of a piece of code sc is the set of all labels in the code
- A piece of code is wellformed iff the labels of all of its instructions are different

Natural semantics of SPUSH

- We want to give natural semantics for the language, on which to base the Hoare logic and type systems.

Natural semantics of SPUSH

- We want to give natural semantics for the language, on which to base the Hoare logic and type systems.
- Need to distinguish between non-termination and errors

Natural semantics of SPUSH

- We want to give natural semantics for the language, on which to base the Hoare logic and type systems.
- Need to distinguish between non-termination and errors
  - We introduce a special abnormal evaluation relation.

Natural semantics of SPUSH

- We want to give natural semantics for the language, on which to base the Hoare logic and type systems.
- Need to distinguish between non-termination and errors
  - We introduce a special abnormal evaluation relation.
  - (Alternatively, the evaluation relation could be indexed by a doubleton)

Natural semantics rules

$$\frac{}{(\ell, \zeta, \sigma) \succ (\ell, \mathsf{load}\ x) \rightarrow (\ell + 1, \sigma(x) :: \zeta, \sigma)}$$

Natural semantics rules

$$\frac{}{(\ell, \zeta, \sigma) \succ (\ell, \text{load } x) \rightarrow (\ell + 1, \sigma(x) :: \zeta, \sigma)}$$

$$\frac{n \in \mathbb{Z}}{(\ell, n :: \zeta, \sigma) \succ (\ell, \text{store } x) \rightarrow (\ell + 1, \zeta, \sigma[x \mapsto n])}$$

Natural semantics rules

$$\frac{}{(\ell, \zeta, \sigma) \succ (\ell, \mathsf{load}\ x) \rightarrow (\ell + 1, \sigma(x) :: \zeta, \sigma)}$$

$$\frac{n \in \mathbb{Z}}{(\ell, n :: \zeta, \sigma) \succ (\ell, \mathsf{store}\ x) \rightarrow (\ell + 1, \zeta, \sigma[x \mapsto n])}$$

$$\frac{\forall n \in \mathbb{Z}, \zeta' \in (\mathbb{Z} \cup \mathbb{B})^*. \zeta \neq n :: \zeta'}{(\ell, \zeta, \sigma) \succ (\ell, \mathsf{store}\ x) \rightarrow\!\!\!\mid (\ell, \zeta, \sigma)}$$

Natural semantics rules

$$\frac{}{(\ell, \zeta, \sigma) \succ (\ell, \text{load } x) \rightarrow (\ell + 1, \sigma(x) :: \zeta, \sigma)}$$

$$\frac{n \in \mathbb{Z}}{(\ell, n :: \zeta, \sigma) \succ (\ell, \text{store } x) \rightarrow (\ell + 1, \zeta, \sigma[x \mapsto n])}$$

$$\frac{\forall n \in \mathbb{Z}, \zeta' \in (\mathbb{Z} \cup \mathbb{B})^*.\, \zeta \neq n :: \zeta'}{(\ell, \zeta, \sigma) \succ (\ell, \text{store } x) \rightarrow\!\!\mid (\ell, \zeta, \sigma)}$$

$$\ldots$$

$$\frac{\ell \in \mathrm{dom}(sc_i) \quad (\ell, \zeta, \sigma) \succ sc_i \rightarrow (\ell'', \zeta'', \sigma'') \quad (\ell'', \zeta'', \sigma'') \succ sc_0 \oplus sc_1 \rightarrow (\ell', \zeta', \sigma')}{(\ell, \zeta, \sigma) \succ sc_0 \oplus sc_1 \rightarrow (\ell', \zeta', \sigma')}$$

Natural semantics rules

$$\frac{}{(\ell, \zeta, \sigma) \succ (\ell, \text{load } x) \rightarrow (\ell + 1, \sigma(x) :: \zeta, \sigma)}$$

$$\frac{n \in \mathbb{Z}}{(\ell, n :: \zeta, \sigma) \succ (\ell, \text{store } x) \rightarrow (\ell + 1, \zeta, \sigma[x \mapsto n])}$$

$$\frac{\forall n \in \mathbb{Z}, \zeta' \in (\mathbb{Z} \cup \mathbb{B})^*. \zeta \neq n :: \zeta'}{(\ell, \zeta, \sigma) \succ (\ell, \text{store } x) \rightarrow\!\!\mid (\ell, \zeta, \sigma)}$$

$$\ldots$$

$$\frac{\ell \in \mathrm{dom}(sc_i) \quad (\ell, \zeta, \sigma) \succ sc_i \rightarrow (\ell'', \zeta'', \sigma'') \quad (\ell'', \zeta'', \sigma'') \succ sc_0 \oplus sc_1 \rightarrow (\ell', \zeta', \sigma')}{(\ell, \zeta, \sigma) \succ sc_0 \oplus sc_1 \rightarrow (\ell', \zeta', \sigma')}$$

$$\frac{\ell \in \mathrm{dom}(sc_i) \quad (\ell, \zeta, \sigma) \succ sc_i \rightarrow\!\!\mid (\ell', \zeta', \sigma')}{(\ell, \zeta, \sigma) \succ sc_0 \oplus sc_1 \rightarrow\!\!\mid (\ell', \zeta', \sigma')}$$

$$\frac{\ell \in \mathrm{dom}(sc_i) \quad (\ell, \zeta, \sigma) \succ sc_i \rightarrow (\ell'', \zeta'', \sigma'') \quad (\ell'', \zeta'', \sigma'') \succ sc_0 \oplus sc_1 \rightarrow\!\!\mid (\ell', \zeta', \sigma')}{(\ell, \zeta, \sigma) \succ sc_0 \oplus sc_1 \rightarrow\!\!\mid (\ell', \zeta', \sigma')}$$

$$\ldots$$

Properties of the semantics

- The natural semantics agrees to the small-step semantics.

Properties of the semantics

- The natural semantics agrees to the small-step semantics.
- Normal termination guarantees the pc to be outside of the domain of the code in the final state.

Properties of the semantics

- The natural semantics agrees to the small-step semantics.
- Normal termination guarantees the pc to be outside of the domain of the code in the final state.
- Abnormal termination guarantees the pc to be in the domain of the code in the final state.

Properties of the semantics

- The natural semantics agrees to the small-step semantics.
- Normal termination guarantees the pc to be outside of the domain of the code in the final state.
- Abnormal termination guarantees the pc to be in the domain of the code in the final state.
- The semantics of a structured piece of code does not depend on the way it is structured

From natural semantics to Hoare logic

- Hoare triples relate pre- and postconditions about a state.

From natural semantics to Hoare logic

- Hoare triples relate pre- and postconditions about a state.
- State contains a pc value and a stack; we use individual constants *pc* and *st* to refer to them in assertions.

From natural semantics to Hoare logic

- Hoare triples relate pre- and postconditions about a state.
- State contains a pc value and a stack; we use individual constants *pc* and *st* to refer to them in assertions.
- The logic we define is an error-free partial-correctness logic.

Hoare rules

$$\frac{}{\left\{\begin{array}{l} \quad (pc = \ell \wedge Q[pc, st \mapsto \ell + 1, x :: st]) \\ \vee (pc \neq \ell \wedge Q) \end{array}\right\} (\ell, \text{load } x) \; \{ Q \}}$$

Hoare rules

$$\frac{}{\left\{\begin{array}{l} \quad (pc = \ell \wedge Q[pc, st \mapsto \ell + 1, x :: st]) \\ \vee (pc \neq \ell \wedge Q) \end{array}\right\} \; (\ell, \mathrm{load}\ x) \; \{\, Q \,\}}$$

Example

$$\frac{}{\{pc = 1 \wedge head(x :: st) = 5\} \; (1, \mathrm{load}\ x) \; \{head(st) = 5\}}$$

Hoare rules

$$\frac{}{\left\{ \begin{array}{l} \quad (pc = \ell \wedge Q[pc, st \mapsto \ell + 1, x :: st]) \\ \vee (pc \neq \ell \wedge Q) \end{array} \right\} (\ell, \mathrm{load}\ x) \ \{ \ Q \ \}}$$

Example

$$\frac{}{\{pc = 1 \wedge x = 5\} \, (1, \mathrm{load}\ x) \, \{head(st) = 5\}}$$

Hoare rules

$$\overline{\left\{\begin{array}{l} \quad (pc = \ell \wedge Q[pc, st \mapsto \ell + 1, x :: st]) \\ \vee (pc \neq \ell \wedge Q) \end{array}\right\} (\ell, \text{load } x) \; \{\, Q \,\}}$$

Hoare rules

$$\frac{}{\left\{\begin{array}{l}\quad(pc = \ell \wedge Q[pc, st \mapsto \ell+1, x :: st]) \\ \vee(pc \neq \ell \wedge Q)\end{array}\right\} (\ell, \text{load } x) \; \{\, Q \,\}}$$

$$\frac{}{\left\{\begin{array}{l}\quad(pc = \ell \wedge \exists z \in \mathbb{Z}, w \in (\mathbb{Z} \cup \mathbb{B})^*. \\ \qquad\qquad st = z :: w \wedge Q[pc, st, x \mapsto \ell+1, w, z]) \\ \vee(pc \neq \ell \wedge Q)\end{array}\right\} (\ell, \text{store } x) \; \{\, Q \,\}}$$

Hoare rules

$$\overline{\left\{\begin{array}{l}\ \ (pc = \ell \wedge Q[pc, st \mapsto \ell + 1, x :: st]) \\ \vee(pc \neq \ell \wedge Q)\end{array}\right\} (\ell, \mathsf{load}\ x)\ \{\ Q\ \}}$$

$$\overline{\left\{\begin{array}{l}\ \ (pc = \ell \wedge \exists z \in \mathbb{Z}, w \in (\mathbb{Z} \cup \mathbb{B})^*. \\ \qquad\qquad\qquad st = z :: w \wedge Q[pc, st, x \mapsto \ell + 1, w, z]) \\ \vee(pc \neq \ell \wedge Q)\end{array}\right\} (\ell, \mathsf{store}\ x)\ \{\ Q\ \}}$$

$$\ldots$$

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

$$\ldots$$

Properties of the logic

Theorem (Soundness of Hoare logic)

If $\{P\}\, sc\, \{Q\}$ *and* $(\ell, \zeta, \sigma) \models_\alpha P$*, then*

- *for any* (ℓ', ζ', σ') *such that* $(\ell, \zeta, \sigma) \succ sc \rightarrow (\ell', \zeta', \sigma')$*, we have* $(\ell', \zeta', \sigma') \models_\alpha Q$
- *and (ii) there is no* (ℓ', ζ', σ') *such that* $(\ell, \zeta, \sigma) \succ sc \rightarrow\!\!| (\ell', \zeta', \sigma')$.

Theorem (Completeness of Hoare logic)

If, for any (ℓ, ζ, σ) *and* α *such that* $(\ell, \zeta, \sigma) \models_\alpha P$*, it holds that*

- *for any* (ℓ', ζ', σ') *such that* $(\ell, \zeta, \sigma) \succ sc \rightarrow (\ell', \zeta', \sigma')$*, we have* $(\ell', \zeta', \sigma') \models_\alpha Q$
- *there is no* (ℓ', ζ', σ') *such that* $(\ell, \zeta, \sigma) \succ sc \rightarrow\!\!| (\ell', \zeta', \sigma')$

then $\{P\}\, sc\, \{Q\}$.

From logic to type systems

- The logic can be weakened to a type system for establishing basic code safety - absence of type and stack underflow errors.

From logic to type systems

- The logic can be weakened to a type system for establishing basic code safety - absence of type and stack underflow errors.
- Intuitive meaning of a typing: if a given piece of code is run from an initial state in a given pretype, then
  - if it terminates normally, the final state is in the posttype
  - it cannot terminate abnormally.

Type system for SPUSH for error-freedom

- Value types $\tau \in$ **ValType** and stack types $\Psi \in$ **StackType** are defined by the grammars

$$\tau ::= \bot \mid \mathrm{int} \mid \mathrm{bool} \mid ?$$
$$\Psi ::= \bot \mid [] \mid \tau :: \Psi \mid *$$

Type system for SPush for error-freedom

- Value types $\tau \in$ **ValType** and stack types $\Psi \in$ **StackType** are defined by the grammars

$$\tau ::= \bot \mid \mathrm{int} \mid \mathrm{bool} \mid ?$$
$$\Psi ::= \bot \mid [] \mid \tau :: \Psi \mid *$$

- A state type $\Pi \in$ **StateType** is a finite set of labelled stack types.

Type system for SPush for error-freedom

- Value types $\tau \in$ **ValType** and stack types $\Psi \in$ **StackType** are defined by the grammars

$$\begin{array}{rcl} \tau & ::= & \perp \mid \mathrm{int} \mid \mathrm{bool} \mid ? \\ \Psi & ::= & \perp \mid [] \mid \tau :: \Psi \mid * \end{array}$$

- A state type $\Pi \in$ **StateType** is a finite set of labelled stack types.
- A state type Π is wellformed iff no label in it labels more than one stack type

Type system for SPUSH for error-freedom

- Value types $\tau \in$ **ValType** and stack types $\Psi \in$ **StackType** are defined by the grammars

$$\tau ::= \bot \mid \text{int} \mid \text{bool} \mid ?$$
$$\Psi ::= \bot \mid [] \mid \tau :: \Psi \mid *$$

- A state type $\Pi \in$ **StateType** is a finite set of labelled stack types.
- A state type Π is wellformed iff no label in it labels more than one stack type
- We will use the notation $\Pi\restriction_L$ for the restriction of a state type Π to a domain $L \subseteq$ **Label**, i.e., $\Pi\restriction_L =_{\text{df}} \{(\ell, \Psi) \mid (\ell, \Psi) \in \Pi, \ell \in L\}$.

Meaning of types

The meanings of value, stack and state types are set-theoretic:

Meaning of types

The meanings of value, stack and state types are set-theoretic:

$$(\!| \bot |\!) =_{\mathrm{df}} \emptyset$$

Meaning of types

The meanings of value, stack and state types are set-theoretic:

$$
\begin{aligned}
(\!|\, \bot \,|\!) &=_{\mathrm{df}} \emptyset \\
(\!|\, \mathrm{int} \,|\!) &=_{\mathrm{df}} \{\mathrm{int}\}
\end{aligned}
$$

Meaning of types

The meanings of value, stack and state types are set-theoretic:

$$\begin{aligned} (\!| \perp |\!) &=_{\mathrm{df}} \emptyset \\ (\!| \mathrm{int} |\!) &=_{\mathrm{df}} \{\mathrm{int}\} \\ (\!| \mathrm{bool} |\!) &=_{\mathrm{df}} \{\mathrm{bool}\} \end{aligned}$$

Meaning of types

The meanings of value, stack and state types are set-theoretic:

$$
\begin{aligned}
(\!|\, \bot \,|\!) &=_{\mathrm{df}} \emptyset \\
(\!|\, \mathrm{int} \,|\!) &=_{\mathrm{df}} \{\mathrm{int}\} \\
(\!|\, \mathrm{bool} \,|\!) &=_{\mathrm{df}} \{\mathrm{bool}\} \\
(\!|\, ? \,|\!) &=_{\mathrm{df}} \{\mathrm{int}, \mathrm{bool}\}
\end{aligned}
$$

Meaning of types

The meanings of value, stack and state types are set-theoretic:

$$
\begin{aligned}
(\!|\, \bot \,|\!) &=_{\mathrm{df}} \emptyset \\
(\!|\, \mathrm{int} \,|\!) &=_{\mathrm{df}} \{\mathrm{int}\} \\
(\!|\, \mathrm{bool} \,|\!) &=_{\mathrm{df}} \{\mathrm{bool}\} \\
(\!|\, ? \,|\!) &=_{\mathrm{df}} \{\mathrm{int}, \mathrm{bool}\} \\
(\!|\, [] \,|\!) &=_{\mathrm{df}} \{[]\}
\end{aligned}
$$

Meaning of types

The meanings of value, stack and state types are set-theoretic:

$$
\begin{aligned}
(\!|\, \bot \,|\!) &=_{\mathrm{df}} \emptyset \\
(\!|\, \mathrm{int} \,|\!) &=_{\mathrm{df}} \{\mathrm{int}\} \\
(\!|\, \mathrm{bool} \,|\!) &=_{\mathrm{df}} \{\mathrm{bool}\} \\
(\!|\, ? \,|\!) &=_{\mathrm{df}} \{\mathrm{int}, \mathrm{bool}\} \\
(\!|\, [] \,|\!) &=_{\mathrm{df}} \{[]\} \\
(\!|\, \tau :: \Psi \,|\!) &=_{\mathrm{df}} \{\delta :: \psi \mid \delta \in (\!|\, \tau \,|\!), \psi \in (\!|\, \Psi \,|\!)\}
\end{aligned}
$$

Meaning of types

The meanings of value, stack and state types are set-theoretic:

$$
\begin{aligned}
(\!|\, \bot \,|\!) &=_{\mathrm{df}} \emptyset \\
(\!|\, \mathrm{int} \,|\!) &=_{\mathrm{df}} \{\mathrm{int}\} \\
(\!|\, \mathrm{bool} \,|\!) &=_{\mathrm{df}} \{\mathrm{bool}\} \\
(\!|\, ? \,|\!) &=_{\mathrm{df}} \{\mathrm{int}, \mathrm{bool}\} \\
(\!|\, [] \,|\!) &=_{\mathrm{df}} \{[]\} \\
(\!|\, \tau :: \Psi \,|\!) &=_{\mathrm{df}} \{\delta :: \psi \mid \delta \in (\!|\, \tau \,|\!), \psi \in (\!|\, \Psi \,|\!)\} \\
(\!|\, * \,|\!) &=_{\mathrm{df}} \{\mathrm{int}, \mathrm{bool}\}^*
\end{aligned}
$$

Meaning of types

The meanings of value, stack and state types are set-theoretic:

$$
\begin{aligned}
(\!|\, \bot \,|\!) &=_{\mathrm{df}} \emptyset \\
(\!|\, \mathrm{int} \,|\!) &=_{\mathrm{df}} \{\mathrm{int}\} \\
(\!|\, \mathrm{bool} \,|\!) &=_{\mathrm{df}} \{\mathrm{bool}\} \\
(\!|\, ? \,|\!) &=_{\mathrm{df}} \{\mathrm{int}, \mathrm{bool}\} \\
(\!|\, [] \,|\!) &=_{\mathrm{df}} \{[]\} \\
(\!|\, \tau :: \Psi \,|\!) &=_{\mathrm{df}} \{\delta :: \psi \mid \delta \in (\!|\, \tau \,|\!), \psi \in (\!|\, \Psi \,|\!)\} \\
(\!|\, * \,|\!) &=_{\mathrm{df}} \{\mathrm{int}, \mathrm{bool}\}^* \\
(\!|\, \Pi \,|\!) &=_{\mathrm{df}} \{(\ell, \psi) \mid (\ell, \Psi) \in \Pi, \psi \in (\!|\, \Psi \,|\!)\}
\end{aligned}
$$

Subtyping rules

$$\frac{}{\tau \leq \tau} \qquad \frac{}{\bot \leq \tau} \qquad \frac{}{\tau \leq \mathsf{?}}$$

Subtyping rules

$$\frac{}{\tau \leq \tau} \qquad \frac{}{\bot \leq \tau} \qquad \frac{}{\tau \leq ?}$$

$$\frac{}{\Psi \leq \Psi} \qquad \frac{\Psi \leq \Psi'' \quad \Psi'' \leq \Psi'}{\Psi \leq \Psi'} \qquad \frac{}{\bot :: \Psi \leq \bot} \qquad \frac{}{\tau :: \bot \leq \bot}$$

Subtyping rules

$$\frac{}{\tau \leq \tau} \qquad \frac{}{\bot \leq \tau} \qquad \frac{}{\tau \leq ?}$$

$$\frac{}{\Psi \leq \Psi} \qquad \frac{\Psi \leq \Psi'' \quad \Psi'' \leq \Psi'}{\Psi \leq \Psi'} \qquad \frac{}{\bot :: \Psi \leq \bot} \qquad \frac{}{\tau :: \bot \leq \bot}$$

$$\frac{}{\bot \leq \Psi} \qquad \frac{}{\Psi \leq *} \qquad \frac{\tau \leq \tau' \quad \Psi \leq \Psi'}{\tau :: \Psi \leq \tau' :: \Psi'}$$

Subtyping rules

$$\frac{}{\tau \leq \tau} \qquad \frac{}{\bot \leq \tau} \qquad \frac{}{\tau \leq ?}$$

$$\frac{}{\Psi \leq \Psi} \qquad \frac{\Psi \leq \Psi'' \quad \Psi'' \leq \Psi'}{\Psi \leq \Psi'} \qquad \frac{}{\bot :: \Psi \leq \bot} \qquad \frac{}{\tau :: \bot \leq \bot}$$

$$\frac{}{\bot \leq \Psi} \qquad \frac{}{\Psi \leq *} \qquad \frac{\tau \leq \tau' \quad \Psi \leq \Psi'}{\tau :: \Psi \leq \tau' :: \Psi'}$$

$$\frac{\forall \ell, \Psi.\, (\ell, \Psi) \in \Pi \supset \Psi = \bot \vee \exists \Psi'.\, (\ell, \Psi') \in \Pi' \wedge \Psi \leq \Psi'}{\Pi \leq \Pi'}$$

Typing rules

$$\frac{}{(\ell, \mathsf{load}\, x) : \begin{array}{c} \{(\ell, \Psi) \mid (\ell+1, \tau :: \Psi) \in \Pi, \mathrm{int} \leq \tau\} \\ \cup\ \{(\ell, *) \mid (\ell+1, *) \in \Pi \cup \Pi\restriction_{\overline{\{\ell\}}}\} \end{array} \longrightarrow \Pi}$$

Typing rules

$$\frac{}{(\ell, \mathsf{load}\ x) : \begin{array}{c}\{(\ell, \Psi) \mid (\ell+1, \tau :: \Psi) \in \Pi, \mathrm{int} \leq \tau\} \\ \cup\ \{(\ell, *) \mid (\ell+1, *) \in \Pi \cup \Pi\restriction_{\overline{\{\ell\}}}\end{array} \longrightarrow \Pi}$$

$$\frac{}{(\ell, \mathsf{store}\ x) : \{(\ell, \mathrm{int} :: \Psi) \mid (\ell+1, \Psi) \in \Pi\} \cup \Pi\restriction_{\overline{\{\ell\}}} \longrightarrow \Pi}$$

Typing rules

$$\frac{}{(\ell, \mathsf{load}\ x) : \begin{array}{c}\{(\ell, \Psi) \mid (\ell+1, \tau :: \Psi) \in \Pi, \mathrm{int} \le \tau\} \\ \cup\ \{(\ell, *) \mid (\ell+1, *) \in \Pi \cup \Pi\restriction_{\overline{\{\ell\}}}\end{array} \longrightarrow \Pi}$$

$$\frac{}{(\ell, \mathsf{store}\ x) : \{(\ell, \mathrm{int} :: \Psi) \mid (\ell+1, \Psi) \in \Pi\} \cup \Pi\restriction_{\overline{\{\ell\}}} \longrightarrow \Pi}$$

$$\ldots$$

$$\frac{sc_0 : \Pi\restriction_{\mathrm{dom}(sc_0)} \longrightarrow \Pi \quad sc_1 : \Pi\restriction_{\mathrm{dom}(sc_1)} \longrightarrow \Pi}{sc_0 \oplus sc_1 : \Pi \longrightarrow \Pi\restriction_{\overline{\mathrm{dom}(sc_0) \cup \mathrm{dom}(sc_1)}}}$$

Typing rules

$$\frac{}{(\ell, \mathsf{load}\ x) : \begin{array}{c}\{(\ell, \Psi) \mid (\ell + 1, \tau :: \Psi) \in \Pi, \mathrm{int} \leq \tau\} \\ \cup\ \{(\ell, *) \mid (\ell + 1, *) \in \Pi \cup \Pi\restriction_{\overline{\{\ell\}}}\end{array} \longrightarrow \Pi}$$

$$\frac{}{(\ell, \mathsf{store}\ x) : \{(\ell, \mathrm{int} :: \Psi) \mid (\ell + 1, \Psi) \in \Pi\} \cup \Pi\restriction_{\overline{\{\ell\}}} \longrightarrow \Pi}$$

$$\dots$$

$$\frac{sc_0 : \Pi\restriction_{\mathrm{dom}(sc_0)} \longrightarrow \Pi \quad sc_1 : \Pi\restriction_{\mathrm{dom}(sc_1)} \longrightarrow \Pi}{sc_0 \oplus sc_1 : \Pi \longrightarrow \Pi\restriction_{\overline{\mathrm{dom}(sc_0) \cup \mathrm{dom}(sc_1)}}}$$

$$\frac{\Pi'_0 \leq \Pi_0 \quad sc : \Pi_0 \longrightarrow \Pi_1 \quad \Pi_1 \leq \Pi'_1}{sc : \Pi'_0 \longrightarrow \Pi'_1}$$

- The type system is sound wrt the natural semantics, but it isn't (cannot) be complete.

Abstract natural semantics

- The type system is sound wrt the natural semantics, but it isn't (cannot) be complete.

Example

1 push *false*
2 gotoF 4
3 store *x*

Abstract natural semantics

- The type system is sound wrt the natural semantics, but it isn't (cannot) be complete.
  - We define *abstract* natural semantics to show the completeness of the type system
  - The abstract semantics is a straightforward rewrite of the concrete semantics to work on abstract states

Abstract natural semantics of SPush

- Abstract states $(\ell, \psi) \in$ **AbsState** are pairs of labels and abstract stacks: **AbsState** $=_{\mathrm{df}}$ **Label** $\times$ **AbsStack**.
  Abstract stack is a stack of (names of) value types: **AbsStack** $=_{\mathrm{df}} \{\mathrm{int}, \mathrm{bool}\}^*$.
- Abstract natural semantics rules:

$$\frac{}{(\ell, \psi) \succ (\ell, \textsf{load } x) \to (\ell + 1, \mathrm{int} :: \psi)}$$

$$\frac{}{(\ell, \mathrm{int} :: \psi) \succ (\ell, \textsf{store } x) \to (\ell + 1, \psi)}$$

$$\frac{\forall \psi' \in \{\mathrm{int}, \mathrm{bool}\}^*.\ \psi \neq \mathrm{int} :: \psi'}{(\ell, \psi) \succ (\ell, \textsf{store } x) \rightarrowtail (\ell, \psi)}$$

$$\ldots$$

Properties of the type system

- Typing is sound wrt the concrete and abstract natural semantics.
- Typing is complete wrt abstract natural semantics.

Properties of the type system

- Typing is sound wrt the concrete and abstract natural semantics.
- Typing is complete wrt abstract natural semantics.

- Besides type systems for stack and type error freedom, compositional type systems presenting data flow analysis can also be devised

Type system for secure information flow

- An example - type system for secure information flow

Type system for secure information flow

- An example - type system for secure information flow
- Central for the type system for secure information flow is a distributive lattice $(\mathrm{D}, \leq, \wedge, \vee, \mathrm{L}, \mathrm{H})$ of security levels for information flowing in the program

Type system for secure information flow

- An example - type system for secure information flow
- Central for the type system for secure information flow is a distributive lattice $(\mathrm{D}, \leq, \wedge, \vee, \mathrm{L}, \mathrm{H})$ of security levels for information flowing in the program
- Abstract states are quadruples of a label $\ell \in$ **Label**, a security level *d* $\in \mathrm{D}$ for the current pc value, and an abstract stack and an abstract store:
  AbsState $=_{\mathrm{df}}$ **Label** $\times\ \mathrm{D}\ \times$ **AbsStack** $\times$ **AbsStore**.

Type system for secure information flow

- An example - type system for secure information flow
- Central for the type system for secure information flow is a distributive lattice $(\mathrm{D}, \leq, \wedge, \vee, \mathrm{L}, \mathrm{H})$ of security levels for information flowing in the program
- Abstract states are quadruples of a label $\ell \in$ **Label**, a security level *d* $\in \mathrm{D}$ for the current pc value, and an abstract stack and an abstract store:
  AbsState $=_{\mathrm{df}}$ **Label** $\times\ \mathrm{D} \times$ **AbsStack** $\times$ **AbsStore**.
- An abstract stack is a list of security levels. An abstract store records the security levels of the variables.

Abstract natural semantics fo secure information flow

$$\frac{}{(\ell, d, \psi, \Sigma) \succ (\ell, \mathsf{load}\ x) \to (\ell + 1, d, \Sigma(x) \vee d :: \psi, \Sigma)} \ \mathrm{load}_{\mathrm{ans}}$$

$$\frac{}{(\ell, d, d' :: \psi, \Sigma) \succ (\ell, \mathsf{store}\ x) \to (\ell + 1, d, \psi, \Sigma[x \mapsto d' \vee d])} \ \mathrm{store}_{\mathrm{ans}}$$

...

$$\frac{\ell \in \mathrm{dom}(sc_i) \qquad (\ell, d, \psi, \Sigma) \succ sc_i \to (\ell'', d'', \psi'', \Sigma'') \\ (\ell'', d'', \psi'', \Sigma'') \succ sc_0 \oplus sc_1 \to (\ell', d', \psi', \Sigma') \qquad sc_0 \oplus sc_1\ \text{single-exit}}{(\ell, d, \psi, \Sigma) \succ sc_0 \oplus sc_1 \to (\ell', d, \psi', \Sigma')} \ \oplus_{\mathrm{ans}}$$

$$\frac{\ell \in \mathrm{dom}(sc_i) \qquad (\ell, d, \psi, \Sigma) \succ sc_i \to (\ell'', d'', \psi'', \Sigma'') \\ (\ell'', d'', \psi'', \Sigma'') \succ sc_0 \oplus sc_1 \to (\ell', d', \psi', \Sigma') \qquad sc_0 \oplus sc_1\ \text{multiple-exit}}{(\ell, d, \psi, \Sigma) \succ sc_0 \oplus sc_1 \to (\ell', d', \psi', \Sigma')} \ \oplus_{\mathrm{ans}}$$

...

Types system for secure information flow

$$\frac{}{(\ell, \text{store } x) : \begin{array}{l}\{(\ell, \Sigma(x) \wedge d, \Sigma(x) :: \Psi, \Sigma) \mid (\ell+1, d, \Psi, \Sigma) \in \Pi\} \\ \cup\ \Pi{\restriction_{\overline{\{\ell\}}}}\end{array} \longrightarrow \Pi}\ \text{store}_{\text{ts}}$$

...

$$\frac{sc_0 : \Pi{\restriction_{\text{dom}(sc_0)}} \longrightarrow \Pi \quad sc_1 : \Pi{\restriction_{\text{dom}(sc_1)}} \longrightarrow \Pi \quad sc_0 \oplus sc_1 \text{ multiple-exit}}{sc_0 \oplus sc_1 : \Pi \longrightarrow \Pi{\restriction_{\overline{\text{dom}(sc_0 \oplus sc_1)}}}}\ \oplus_{\text{ts}}$$

Conclusion

- The original idea of structuring low level languages to obtain a compositional Hoare logic applies to stack based languages.
- The logic can be weakened to type system attesting code safety, but also to type systems reflecting dataflow analysis.
- Abnormal termination can be handled without a problem.