Lecture #14: Proving Type-Safety

CS 6371: Advanced Programming Languages

February 27, 2020

A language's static semantics are designed to eliminate runtime *stuck states*. Recall that stuck states are non-final configurations whose runtime behavior is undefined according to the language's small-step operational semantics.

Definition 1 (Final state). Configuration $\langle c, \sigma \rangle$ is a *final state* if $c = \texttt{skip}$.

Definition 2 (Stuck state). Configuration $\langle c, \sigma \rangle$ is a *stuck state* if it is non-final and there is no configuration $\langle c', \sigma' \rangle$ such that the judgment $\langle c, \sigma \rangle \rightarrow_1 \langle c', \sigma' \rangle$ is derivable.

A language is said to be *type-safe* if its static semantics prevent all stuck states. To define this formally, we need a definition of what it means for a program to be *well-typed*.

Definition 3 (Well-typed commands). A program c is *well-typed in context* Γ if there exists a typing context Γ' such that judgment $\Gamma \vdash c : \Gamma'$ is derivable. We say simply that c is *well-typed* if it is well-typed in context $\perp$.

Definition 4 (Well-typed expressions). Similarly, an expression e is *well-typed in context* Γ if there exists a type τ such that judgment $\Gamma \vdash e : \tau$ is derivable; and we say simply that e is *well-typed* if it is well-typed in context $\perp$.

Definition 5 (Type safety). A language is *type-safe* if for every well-typed program c and number of steps $n \in \mathbb{N}$, if $\langle c, \perp \rangle \rightarrow_n \langle c', \sigma' \rangle$ then configuration $\langle c', \sigma' \rangle$ is not a stuck state.

Proving that a language is type-safe can be a non-trivial undertaking. To do so, one usually must first define a notion of consistency between a typing context and a store. This allows us to say that the typing context is consistent with each memory state that the program enters at runtime.

Definition 6 (Modeling of stores by typing contexts). We say that a typing context Γ *models* a store σ, writing $\Gamma \models \sigma$, if for all $v \in \Gamma^{\leftarrow}$, the following two conditions hold: (1) If $\Gamma(v) = (\mathit{int}, T)$ then $\sigma(v) \in \mathbb{Z}$, and (2) if $\Gamma(v) = (\mathit{bool}, T)$ then $\sigma(v) \in \{T, F\}$.

Another complication involves variables appearing and disappearing from the typing context due to local scopes (e.g., `if` statements). To type-check the intermediate steps introduced by such scopes, we will need the concept of a typing context *stack*—a non-empty list of typing contexts, $\Gamma_1, \ldots, \Gamma_n$ $(n \geq 1)$ such that $\Gamma_1 \succeq \cdots \succeq \Gamma_n$, where $\preceq$ is the following subtyping relation.

Definition 7 (Subtypes)**.** Typing context Γ_1 is a *subtype* of typing context Γ_2, written $\Gamma_1 \preceq \Gamma_2$, if $\forall v \in \Gamma_2^{\leftarrow} \; . \; (\Gamma_2(v) = (\tau, p)) \Rightarrow \big((\Gamma_1(v) = (\tau, q)) \wedge (p \Rightarrow q)\big)$.

Subtyping relation $\Gamma_1 \preceq \Gamma_2$ asserts that context Γ_1 is "less restrictive" than Γ_2 in the sense that it might have more variables in it or it might assert that variables in Γ_2 are now initialized and can therefore be read. Conceptually, the context stack lists the typing contexts for each nested scope, from outermost (most restrictive) to innermost (least restrictive). Henceforth we will abbreviate a context stack by writing a vector arrow above it ($\overrightarrow{\Gamma}$) and we will refer to the first (most restrictive, outermost) context in such a stack by leaving off the arrow (Γ). We will write $\overrightarrow{\Gamma} \models \sigma$ if for every context Γ_i in stack $\overrightarrow{\Gamma}$ we have $\Gamma_i \models \sigma$.

We can keep track of commands contained within a local scope by adding a new command syntax `{`c`}`. The braces are typing annotations; they have no effect upon programs at runtime (see operational semantics Rules S1 and S2). We add them solely to help the type-checker remember the extent of local scopes (see Rules 25, 26, and 27).

It is important to understand why changing the operational and static semantics still allows us to prove type-safety for the original operational and static semantics. The proof argument is as follows: (1) It is trivial to prove that the new operational semantics (with braces) is *equivalent* to the old operational semantics (without braces). This is because the braces have no effect upon a program's runtime behavior. (2) It is also trivial to prove that if a program in the old language (no braces) is well-typed according to the old static semantics, then it is also well-typed according to the new static semantics. This is because the new static semantics only differ from the old with regard to braces, and the old language has no braces. Thus, if we prove that the new semantics is type-safe (in the sense that the static semantics eliminate all stuck states) then we've proved that the old semantics are also type-safe.

We prove type-safety by proving two lemmas: progress and subject reduction. The progress lemma basically says that if a program is well-typed and is not in a final state, then it can take a step. The subject reduction lemma basically says that if a program takes a step, then the new program it steps to is also well-typed. Together these two lemmas imply that well-typed programs either step forever or eventually enter final states; they never enter stuck states.

Lemma 1 (Progress of expressions)**.** *If* $\Gamma \vdash e : \tau$ *and* $\Gamma \models \sigma$ *both hold, then either* $e = n$, $e =$ `true`, $e =$ `false`, *or there exists* e' *and* σ' *such that* $\langle e, \sigma\rangle \rightarrow_1 \langle e', \sigma'\rangle$ *holds.*

Lemma 2 (Progress of commands)**.** *If* $\overrightarrow{\Gamma} \vdash c : \Gamma'$ *and* $\overrightarrow{\Gamma} \models \sigma$ *both hold, then either* $c =$ `skip` *or there exists* c' *and* σ' *such that* $\langle c, \sigma\rangle \rightarrow_1 \langle c', \sigma'\rangle$ *holds.*

Lemma 3 (Subject reduction of expressions)**.** *If* $\Gamma \vdash e_1 : \tau$ *and* $\Gamma \models \sigma_1$ *and* $\langle e_1, \sigma_1\rangle \rightarrow_1 \langle e_2, \sigma_2\rangle$ *all hold, then* $\Gamma \vdash e_2 : \tau$ *and* $\Gamma \models \sigma_2$ *both hold.*

Lemma 4 (Subject reduction of commands)**.** *If* $\overrightarrow{\Gamma_1} \vdash c_1 : \Gamma'$ *and* $\overrightarrow{\Gamma_1} \models \sigma_1$ *and* $\langle c_1, \sigma_1\rangle \rightarrow_1 \langle c_2, \sigma_2\rangle$ *all hold, then there exists a stack* $\overrightarrow{\Gamma_2}$ *such that* $\overrightarrow{\Gamma_2} \vdash c_2 : \Gamma'$ *and* $\overrightarrow{\Gamma_2} \models \sigma_2$ *and* $\Gamma_2 \preceq \Gamma_1$ *all hold.*

1 Static Semantics of Typed-SIMPL

1.1 Commands

$$\Gamma \vdash \texttt{skip} : \Gamma \tag{1}$$

$$\frac{v \notin \Gamma^{\leftarrow}}{\Gamma \vdash \texttt{int}\ v : \Gamma[v \mapsto (\mathit{int}, F)]} \tag{2}$$

$$\frac{v \notin \Gamma^{\leftarrow}}{\Gamma \vdash \texttt{bool}\ v : \Gamma[v \mapsto (\mathit{bool}, F)]} \tag{3}$$

$$\frac{\overrightarrow{\Gamma} \vdash c_1 : \Gamma_2 \qquad \Gamma_2 \vdash c_2 : \Gamma'}{\overrightarrow{\Gamma} \vdash c_1; c_2 : \Gamma'} \tag{4}$$

$$\frac{\Gamma \vdash e : \tau \qquad \Gamma(v) = (\tau, p)}{\Gamma \vdash v\texttt{:=}e : \Gamma[v \mapsto (\tau, T)]} \tag{5}$$

$$\frac{\Gamma \vdash e : \mathit{bool} \qquad \overrightarrow{\Gamma} \vdash c_1 : \Gamma_1 \qquad \Gamma \vdash c_2 : \Gamma_2}{\overrightarrow{\Gamma} \vdash \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2 : \Gamma} \tag{6}$$

$$\frac{\Gamma \vdash e : \mathit{bool} \qquad \Gamma \vdash c : \Gamma_1}{\Gamma \vdash \texttt{while}\ e\ \texttt{do}\ c : \Gamma} \tag{7}$$

1.2 Expressions

$$\Gamma \vdash n : \mathit{int} \tag{8}$$

$$\Gamma \vdash \texttt{true} : \mathit{bool} \tag{9}$$

$$\Gamma \vdash \texttt{false} : \mathit{bool} \tag{10}$$

$$\frac{\Gamma(v) = (\tau, T)}{\Gamma \vdash v : \tau} \tag{11}$$

$$\frac{\Gamma \vdash e_1 : \mathit{int} \qquad \Gamma \vdash e_2 : \mathit{int}}{\Gamma \vdash e_1\ \mathit{aop}\ e_2 : \mathit{int}} \tag{12}$$

$$\frac{\Gamma \vdash e_1 : \mathit{bool} \qquad \Gamma \vdash e_2 : \mathit{bool}}{\Gamma \vdash e_1\ \mathit{bop}\ e_2 : \mathit{bool}} \tag{13}$$

$$\frac{\Gamma \vdash e_1 : \mathit{int} \qquad \Gamma \vdash e_2 : \mathit{int}}{\Gamma \vdash e_1\texttt{<=}e_2 : \mathit{bool}} \tag{14}$$

$$\frac{\Gamma \vdash e : \mathit{bool}}{\Gamma \vdash \texttt{!}e : \mathit{bool}} \tag{15}$$

$$\frac{\overrightarrow{\Gamma} \vdash c : \Gamma'}{\Gamma_1, \overrightarrow{\Gamma} \vdash \texttt{\{}c\texttt{\}} : \Gamma_1} \tag{\textsc{Scoping}}$$

2 Small-step Operational Semantics of Typed-SIMPL

2.1 Commands

$$\langle \texttt{int}\ v, \sigma\rangle \rightarrow_1 \langle \texttt{skip}, \sigma\rangle \tag{16}$$

$$\langle \texttt{bool}\ v, \sigma\rangle \rightarrow_1 \langle \texttt{skip}, \sigma\rangle \tag{17}$$

$$\frac{\langle c_1, \sigma\rangle \rightarrow_1 \langle c_1', \sigma'\rangle}{\langle c_1; c_2, \sigma\rangle \rightarrow_1 \langle c_1'; c_2, \sigma'\rangle} \tag{18}$$

$$\langle \texttt{skip}; c, \sigma\rangle \rightarrow_1 \langle c, \sigma\rangle \tag{19}$$

$$\frac{\langle e, \sigma\rangle \rightarrow_1 \langle e', \sigma'\rangle}{\langle v\,\texttt{:=}e, \sigma\rangle \rightarrow_1 \langle v\,\texttt{:=}e', \sigma'\rangle} \tag{20}$$

$$\langle v\,\texttt{:=}n, \sigma\rangle \rightarrow_1 \langle \texttt{skip}, \sigma[v \mapsto n]\rangle \tag{21}$$

$$\langle v\,\texttt{:=true}, \sigma\rangle \rightarrow_1 \langle \texttt{skip}, \sigma[v \mapsto T]\rangle \tag{22}$$

$$\langle v\,\texttt{:=false}, \sigma\rangle \rightarrow_1 \langle \texttt{skip}, \sigma[v \mapsto F]\rangle \tag{23}$$

$$\frac{\langle e, \sigma\rangle \rightarrow_1 \langle e', \sigma\rangle}{\langle \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2, \sigma\rangle \rightarrow_1 \langle \texttt{if}\ e'\ \texttt{then}\ c_1\ \texttt{else}\ c_2, \sigma'\rangle} \tag{24}$$

$$\langle \texttt{if true then}\ c_1\ \texttt{else}\ c_2, \sigma\rangle \rightarrow_1 \langle \texttt{\{}c_1\texttt{\}}, \sigma\rangle \tag{25}$$

$$\langle \texttt{if false then}\ c_1\ \texttt{else}\ c_2, \sigma\rangle \rightarrow_1 \langle \texttt{\{}c_2\texttt{\}}, \sigma\rangle \tag{26}$$

$$\langle \texttt{while}\ e\ \texttt{do}\ c, \sigma\rangle \rightarrow_1 \langle \texttt{if}\ e\ \texttt{then}\ (\texttt{\{}c\texttt{\}}; \texttt{while}\ e\ \texttt{do}\ c)\ \texttt{else skip}, \sigma\rangle \tag{27}$$

$$\frac{\langle c, \sigma\rangle \rightarrow_1 \langle c', \sigma'\rangle}{\langle \texttt{\{}c\texttt{\}}, \sigma\rangle \rightarrow_1 \langle \texttt{\{}c'\texttt{\}}, \sigma'\rangle} \tag{S1}$$

$$\langle \texttt{\{skip\}}, \sigma\rangle \rightarrow_1 \langle \texttt{skip}, \sigma\rangle \tag{S2}$$

2.2 Expressions

$$\frac{\sigma(v) = n}{\langle v, \sigma\rangle \rightarrow_1 \langle n, \sigma\rangle} \tag{28}$$

$$\frac{\sigma(v) = T}{\langle v, \sigma\rangle \rightarrow_1 \langle \texttt{true}, \sigma\rangle} \tag{29}$$

$$\frac{\sigma(v) = F}{\langle v, \sigma\rangle \rightarrow_1 \langle \texttt{false}, \sigma\rangle} \tag{30}$$

$$\frac{\langle e_1, \sigma\rangle \rightarrow_1 \langle e_1', \sigma'\rangle \qquad op \in aop \cup bop \cup \{\texttt{<=}\}}{\langle e_1\ op\ e_2, \sigma\rangle \rightarrow_1 \langle e_1'\ op\ e_2, \sigma'\rangle} \tag{31}$$

$$\frac{\langle e_2, \sigma\rangle \rightarrow_1 \langle e_2', \sigma'\rangle \qquad op \in aop \cup \{\texttt{<=}\}}{\langle n\ op\ e_2, \sigma\rangle \rightarrow_1 \langle n\ op\ e_2', \sigma'\rangle} \tag{32}$$

$$\langle n_1 \texttt{+} n_2, \sigma\rangle \rightarrow_1 \langle n_1 + n_2, \sigma\rangle \tag{33}$$

$$\langle n_1 \texttt{-} n_2, \sigma\rangle \rightarrow_1 \langle n_1 - n_2, \sigma\rangle \tag{34}$$

$$\langle n_1 \texttt{*} n_2, \sigma\rangle \rightarrow_1 \langle n_1 n_2, \sigma\rangle \tag{35}$$

$$\frac{n_1 \leq n_2}{\langle n_1 \texttt{<=} n_2, \sigma\rangle \rightarrow_1 \langle \texttt{true}, \sigma\rangle} \tag{36}$$

$$\frac{n_1 > n_2}{\langle n_1 \texttt{<=} n_2, \sigma\rangle \rightarrow_1 \langle \texttt{false}, \sigma\rangle} \tag{37}$$

$$\langle \texttt{true}\ \texttt{\&\&}\ e_2, \sigma\rangle \rightarrow_1 \langle e_2, \sigma\rangle \tag{38}$$

$$\langle \texttt{false}\ \texttt{\&\&}\ e_2, \sigma\rangle \rightarrow_1 \langle \texttt{false}, \sigma\rangle \tag{39}$$

$$\langle \texttt{true}\ \texttt{||}\ e_2, \sigma\rangle \rightarrow_1 \langle \texttt{true}, \sigma\rangle \tag{40}$$

$$\langle \texttt{false}\ \texttt{||}\ e_2, \sigma\rangle \rightarrow_1 \langle e_2, \sigma\rangle \tag{41}$$

$$\frac{\langle e, \sigma\rangle \rightarrow_1 \langle e', \sigma\rangle}{\langle \texttt{!}e, \sigma\rangle \rightarrow_1 \langle \texttt{!}e', \sigma'\rangle} \tag{42}$$

$$\langle \texttt{!true}, \sigma\rangle \rightarrow_1 \langle \texttt{false}, \sigma\rangle \tag{43}$$

$$\langle \texttt{!false}, \sigma\rangle \rightarrow_1 \langle \texttt{true}, \sigma\rangle \tag{44}$$