5.2 Large-step Semantics of SIMPL-F

5.2.1 Commands

$$\langle \texttt{skip}, \sigma\rangle \Downarrow \sigma \tag{1}$$

$$\langle td\ v, \sigma\rangle \Downarrow \sigma \tag{2}$$

$$\frac{\langle c_1, \sigma\rangle \Downarrow \sigma_2 \qquad \langle c_2, \sigma_2\rangle \Downarrow \sigma'}{\langle c_1; c_2, \sigma\rangle \Downarrow \sigma'} \tag{3}$$

$$\frac{\langle e, \sigma\rangle \Downarrow u}{\langle v \,\texttt{:=}\, e, \sigma\rangle \Downarrow \sigma[v \mapsto u]} \tag{4}$$

$$\frac{\langle e, \sigma\rangle \Downarrow T \qquad \langle c_1, \sigma\rangle \Downarrow \sigma'}{\langle \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2, \sigma\rangle \Downarrow \sigma'} \tag{5}$$

$$\frac{\langle e, \sigma\rangle \Downarrow F \qquad \langle c_2, \sigma\rangle \Downarrow \sigma'}{\langle \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2, \sigma\rangle \Downarrow \sigma'} \tag{6}$$

$$\frac{\langle \texttt{if}\ e\ \texttt{then}\ (c; \texttt{while}\ e\ \texttt{do}\ c)\ \texttt{else}\ \texttt{skip}, \sigma\rangle \Downarrow \sigma'}{\langle \texttt{while}\ e\ \texttt{do}\ c, \sigma\rangle \Downarrow \sigma'} \tag{7}$$

5.2.2 Expressions

$$\langle n, \sigma\rangle \Downarrow n \tag{8}$$

$$\langle \texttt{true}, \sigma\rangle \Downarrow T \tag{9}$$

$$\langle \texttt{false}, \sigma\rangle \Downarrow F \tag{10}$$

$$\langle v, \sigma\rangle \Downarrow \sigma(v) \tag{11}$$

$$\frac{\langle e_1, \sigma\rangle \Downarrow n_1 \qquad \langle e_2, \sigma\rangle \Downarrow n_2}{\langle e_1 \texttt{<=} e_2, \sigma\rangle \Downarrow n_1 \le n_2} \tag{12}$$

$$\frac{\langle e_1, \sigma\rangle \Downarrow p \qquad \langle e_2, \sigma\rangle \Downarrow q}{\langle e_1\ \texttt{\&\&}\ e_2, \sigma\rangle \Downarrow p \wedge q} \tag{13}$$

$$\frac{\langle e_1, \sigma\rangle \Downarrow p \qquad \langle e_2, \sigma\rangle \Downarrow q}{\langle e_1\ \texttt{||}\ e_2, \sigma\rangle \Downarrow p \vee q} \tag{14}$$

$$\frac{\langle e, \sigma\rangle \Downarrow p}{\langle \texttt{!}e, \sigma\rangle \Downarrow \neg p} \tag{15}$$

$$\frac{\langle e_1, \sigma\rangle \Downarrow n_1 \qquad \langle e_2, \sigma\rangle \Downarrow n_2}{\langle e_1 \,\texttt{+}\, e_2, \sigma\rangle \Downarrow n_1 + n_2} \tag{16}$$

$$\frac{\langle e_1, \sigma\rangle \Downarrow n_1 \qquad \langle e_2, \sigma\rangle \Downarrow n_2}{\langle e_1 \,\texttt{-}\, e_2, \sigma\rangle \Downarrow n_1 - n_2} \tag{17}$$

$$\frac{\langle e_1, \sigma\rangle \Downarrow n_1 \qquad \langle e_2, \sigma\rangle \Downarrow n_2}{\langle e_1 \,\texttt{*}\, e_2, \sigma\rangle \Downarrow n_1 n_2} \tag{18}$$

$$\langle \texttt{fun}(td_1\ v_1, \ldots, td_n\ v_n)\{c\}, \sigma\rangle \Downarrow \lambda(v_1, \ldots, v_n).c \tag{19}$$

$$\frac{\begin{array}{c}\langle e_0, \sigma\rangle \Downarrow \lambda(v_1, \ldots, v_n).c \qquad \forall i \in 1..n\ .\ \langle e_i, \sigma\rangle \Downarrow u_i \\ \langle c, \sigma[v_1 \mapsto u_1] \cdots [v_n \mapsto u_n]\rangle \Downarrow \sigma'\end{array}}{\langle e_0(e_1, \ldots, e_n), \sigma\rangle \Downarrow \sigma'(\texttt{ret})} \tag{20}$$

5.3 Typing Rules for SIMPL-F

5.3.1 Commands

$$\Gamma \vdash \texttt{skip} : \Gamma \tag{21}$$

$$\frac{v \notin \Gamma^{\leftarrow}}{\Gamma \vdash td\ v : \Gamma[v \mapsto (\mathit{typ}(td), F)]} \tag{22}$$

$$\frac{\Gamma \vdash c_1 : \Gamma_2 \qquad \Gamma_2 \vdash c_2 : \Gamma'}{\Gamma \vdash c_1; c_2 : \Gamma'} \tag{23}$$

$$\frac{\Gamma \vdash e : \tau \qquad \Gamma(v) = (\tau, p)}{\Gamma \vdash v\texttt{:=}e : \Gamma[v \mapsto (\tau, T)]} \tag{24}$$

$$\frac{\Gamma \vdash e : \mathit{bool} \qquad \Gamma \vdash c_1 : \Gamma_1 \qquad \Gamma \vdash c_2 : \Gamma_2}{\Gamma \vdash \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2 : \Gamma} \tag{25}$$

$$\frac{\Gamma \vdash e : \mathit{bool} \qquad \Gamma \vdash c : \Gamma_1}{\Gamma \vdash \texttt{while}\ e\ \texttt{do}\ c : \Gamma} \tag{26}$$

5.3.2 Non-function Expressions

$$\Gamma \vdash n : \mathit{int} \tag{27}$$

$$\Gamma \vdash \texttt{true} : \mathit{bool} \tag{28}$$

$$\Gamma \vdash \texttt{false} : \mathit{bool} \tag{29}$$

$$\frac{\Gamma(v) = (\tau, T)}{\Gamma \vdash v : \tau} \tag{30}$$

$$\frac{\Gamma \vdash e_1 : \mathit{int} \qquad \Gamma \vdash e_2 : \mathit{int}}{\Gamma \vdash e_1\ \mathit{aop}\ e_2 : \mathit{int}} \tag{31}$$

$$\frac{\Gamma \vdash e_1 : \mathit{bool} \qquad \Gamma \vdash e_2 : \mathit{bool}}{\Gamma \vdash e_1\ \mathit{bop}\ e_2 : \mathit{bool}} \tag{32}$$

$$\frac{\Gamma \vdash e_1 : \mathit{int} \qquad \Gamma \vdash e_2 : \mathit{int}}{\Gamma \vdash e_1\texttt{<=}e_2 : \mathit{bool}} \tag{33}$$

$$\frac{\Gamma \vdash e : \mathit{bool}}{\Gamma \vdash \texttt{!}e : \mathit{bool}} \tag{34}$$

5.3.3 Function Expressions

$$\frac{\begin{array}{cc} \forall i \in 1..n\ .\ \tau_i = \mathit{typ}(td_i) & v_1, \ldots, v_n \notin \Gamma^{\leftarrow} \\ \Gamma[v_1 \mapsto (\tau_1, T)] \cdots [v_n \mapsto (\tau_n, T)] \vdash c : \Gamma' & \Gamma'(\texttt{ret}) = (\tau_0, T) \end{array}}{\Gamma \vdash \texttt{fun}(td_1\ v_1, \ldots, td_n\ v_n)\{c\} : (\tau_1 \times \cdots \times \tau_n) \to \tau_0} \tag{35}$$

$$\frac{\Gamma \vdash e_0 : (\tau_1 \times \cdots \times \tau_n) \to \tau_0 \qquad \forall i \in 1..n\ .\ \Gamma \vdash e_i : \tau_i}{\Gamma \vdash e_0(e_1, \ldots, e_n) : \tau_0} \tag{36}$$

Here, *typ* is just a function that turns a type declaration (a sequence of characters typed on your keyboard) into an equivalent type (a mathematical entity). That is, *typ* can be formally defined as:

$$\begin{aligned} \mathit{typ}(\texttt{int}) &= \mathit{int} \\ \mathit{typ}(\texttt{bool}) &= \mathit{bool} \\ \mathit{typ}(\texttt{fun}(td_1 \texttt{*} \cdots \texttt{*} td_n\texttt{->}td_0)) &= (\mathit{typ}(td_1) \times \cdots \times \mathit{typ}(td_n)) \to \mathit{typ}(td_0) \end{aligned}$$

You do not have to implement *typ* because it is already implemented within the parser. (It is the part of the parser that creates OCaml values like `TypInt` that implement SIMPL-F types.)

5 Reference

5.1 Syntax of SIMPL-F

commands	$c ::= \texttt{skip} \mid c_1; c_2 \mid v\texttt{:=}e \mid \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2$
	$\mid \texttt{while}\ e\ \texttt{do}\ c \mid td\ v$
expressions	$e ::= n \mid \texttt{true} \mid \texttt{false} \mid v \mid e_1\ aop\ e_2 \mid e_1\ bop\ e_2 \mid e_1\texttt{<=}e_2 \mid !e$
	$\mid \texttt{fun}(td_1\ v_1, \ldots, td_n\ v_n)\{c\} \mid e_0(e_1, \ldots, e_n)$
type declarations	$td ::= \texttt{int} \mid \texttt{bool} \mid \texttt{fun}(td_1 \texttt{*} \cdots \texttt{*}\, td_n \texttt{->}\, td_0)$
arithmetic operators	$aop ::= \texttt{+} \mid \texttt{-} \mid \texttt{*}$
boolean operators	$bop ::= \texttt{\&\&} \mid \texttt{||}$
variable names	v
integer constants	n
types	$\tau ::= int \mid bool \mid (\tau_1 \times \cdots \times \tau_n) \rightarrow \tau_0$
values	$u \in \mathbb{Z} \cup \{T, F\} \cup \{\lambda(v_1, \ldots, v_n).c\}$
stores	$\sigma : v \rightharpoonup u$
typing contexts	$\Gamma : v \rightharpoonup \tau \times \{T, F\}$