`salt0`: Straight Line Programs

`CS392-M1`: *Rust, In Practice and in Theory*

1 Syntax

The following is the BNF specification for the syntax of `salt0`. We fix an arbitrary set $\mathcal{V}$ of variables.

x	(variables, $\mathcal{V}$)
n	(integers, $\mathbb{Z}$)
$v ::= n \mid \texttt{()}$	(values, Val)
$e ::= n \mid \texttt{()} \mid x \mid e \texttt{ + } e$	(expressions, $\mathcal{E}$)
$s ::= \texttt{let } x \texttt{ = } e$	(statements, $\mathcal{S}$)
$p ::= \{s \texttt{ ;}\}\ e$	(programs, $\mathcal{P}$)
$t ::= \texttt{i32} \mid \texttt{()}$	(types, $\mathcal{T}$)

2 Typing

There are three kinds of typing judgments:

$\Gamma \vdash e : t$	(expressions)
$\Gamma \vdash s \dashv \Gamma$	(statments)
$\Gamma \vdash p : t$	(programs)

The meta-variable Γ stands for a typing context (a.k.a., static environment), which we will take to be a map from variables to types (i.e., a map of the form $\mathcal{V} \mapsto \mathcal{T}$). Note that statements do not have types, but they can affect the state of the context (i.e., they can have side-effects). The following are the typing rules for `salt0`.

$$\frac{n \in \mathbb{Z}}{\Gamma \vdash n : \texttt{i32}}\ \text{(int)} \qquad \frac{}{\Gamma \vdash \texttt{()} : \texttt{()}}\ \text{(unit)} \qquad \frac{(x \mapsto t) \in \Gamma}{\Gamma \vdash x : t}\ \text{(var)}$$

$$\frac{\Gamma \vdash e_1 : \texttt{i32} \qquad \Gamma \vdash e_2 : \texttt{i32}}{\Gamma \vdash e_1 \texttt{ + } e_2 : \texttt{i32}}\ \text{(add)}$$

$$\frac{x \notin \mathsf{dom}(\Gamma) \qquad \Gamma \vdash e : t}{\Gamma \vdash \texttt{let } x \texttt{ = } e \dashv \Gamma[x \mapsto t]}\ \text{(let)}$$

$$\frac{\Gamma_1 \vdash s \dashv \Gamma_2 \qquad \Gamma_2 \vdash p : t}{\Gamma_1 \vdash s \texttt{ ; } p : t}\ \text{(prog)}$$

3 Evaluation

There are three kinds of (big-step) evaluation judgments:

$$\langle\, S\,,\, e\,\rangle \Downarrow v \qquad \text{(expressions)}$$
$$\langle\, S\,,\, s\,\rangle \Downarrow S \qquad \text{(statements)}$$
$$\langle\, S\,,\, p\,\rangle \Downarrow v \qquad \text{(programs)}$$

The meta-variable S stands for a store (a.k.a., dynamic environment), which we will take to be a map from variables to values (i.e., a map of the form $\mathcal{V} \mapsto \mathsf{Val}$). Note that statements do not have values, but they can affect the state of the store. The following are the evaluation rules for `salt0`.

$$\frac{n \in \mathbb{Z}}{\langle\, S\,,\, n\,\rangle \Downarrow n}\ \text{(int)} \qquad \frac{}{\langle\, S\,,\, \texttt{()}\,\rangle \Downarrow \texttt{()}}\ \text{(unit)} \qquad \frac{}{\langle\, S\,,\, x\,\rangle \Downarrow S(x)}\ \text{(var)}$$

$$\frac{\langle\, S\,,\, e_1\,\rangle \Downarrow v_1 \qquad \langle\, S\,,\, e_2\,\rangle \Downarrow v_2}{\langle\, S\,,\, e_1 \texttt{ + } e_2\,\rangle \Downarrow v_1 + v_2}\ \text{(add)}$$

$$\frac{\langle\, S\,,\, e\,\rangle \Downarrow v}{\langle\, S\,,\, \texttt{let } x \texttt{ = } e\,\rangle \Downarrow S[x \mapsto v]}\ \text{(let)}$$

$$\frac{\langle\, S_1\,,\, s\,\rangle \Downarrow S_2 \qquad \langle\, S_2\,,\, p\,\rangle \Downarrow v}{\langle\, S_1\,,\, s \texttt{ ; } p\,\rangle \Downarrow v}\ \text{(prog)}$$

4 Reduction

There are three kinds of reduction judgments:

$$\langle\, S\,,\, e\,\rangle \longrightarrow \langle\, S\,,\, e\,\rangle \qquad \text{(expressions)}$$
$$\langle\, S\,,\, s\,\rangle \longrightarrow \langle\, S\,,\, s\,\rangle \qquad \text{(statements)}$$
$$\langle\, S\,,\, p\,\rangle \longrightarrow \langle\, S\,,\, p\,\rangle \qquad \text{(programs)}$$

We will extend our syntax to include holes and to allow for evaluating statements:

$$s ::= \texttt{let } x \texttt{ = } e \mid \bullet \qquad \text{(statements)}$$
$$e[\![\ ]\!] ::= [\![\ ]\!] \mid e[\![\ ]\!] \texttt{ + } e \mid e \texttt{ + } e[\![\ ]\!] \qquad \text{(holed expressions)}$$
$$s[\![\ ]\!] ::= \texttt{let } x \texttt{ = } e[\![\ ]\!] \qquad \text{(holed statement)}$$
$$p[\![\ ]\!] ::= \{s[\![\ ]\!] \texttt{ ;}\} \mid \{s \texttt{ ;}\}\ e[\![\ ]\!] \qquad \text{(holed programs)}$$

The $\bullet$ statement is a placeholder for statements that have been fully evaluated. The following are the reduction rules for `salt0`.

$$\frac{(x \mapsto v) \in S}{\langle\, S\,,\, x\,\rangle \longrightarrow \langle\, S\,,\, v\,\rangle}\ \text{(var)}$$

$$\frac{}{\langle\, S\,,\, v_1 \texttt{ + } v_2\,\rangle \longrightarrow \langle\, S\,,\, v_1 + v_2\,\rangle}\ \text{(add)}$$

$$\frac{}{\langle\, S\,,\, \texttt{let } x \texttt{ = } v\,\rangle \longrightarrow \langle\, S[x \mapsto v]\,,\, \bullet\,\rangle}\ \text{(let)}$$

$$\frac{\langle\ S_1\ ,\ s_1\ \rangle \longrightarrow \langle\ S_2\ ,\ s_2\ \rangle}{\langle\ S_1\ ,\ s_1\ \texttt{;}\ p\ \rangle \longrightarrow \langle\ S_2\ ,\ s_2\ \texttt{;}\ p\ \rangle}\ (\text{prog}_1)$$

$$\frac{}{\langle\ S\ ,\ \bullet\ \texttt{;}\ p\ \rangle \longrightarrow \langle\ S\ ,\ p\ \rangle}\ (\text{prog}_2)$$

$$\frac{\langle\ S_1\ ,\ e_1\ \rangle \longrightarrow \langle\ S_2\ ,\ e_2\ \rangle}{\langle\ S_1\ ,\ \sigma[\![e_1]\!]\ \rangle \longrightarrow \langle\ S_2\ ,\ \sigma[\![e_2]\!]\ \rangle}\ (\text{hole})$$

5 Meta-Theory

These first two theorems give a correspondence between the evaluation rules and the reduction rules. We will appeal to these theorems to justify using reduction rules to help use visualize the evaluation process.

Theorem. *(Operational Adequacy) For any store S, program p, and value v, if $\langle\ S\ ,\ p\ \rangle \Downarrow v$, then there is store S' such that $\langle\ S\ ,\ p\ \rangle \longrightarrow^\star \langle\ S'\ ,\ v\ \rangle$.*

Theorem. *(Operational Soundness) For any stores S and S', program p, and value v, if $\langle\ S\ ,\ p\ \rangle \longrightarrow^\star \langle\ S'\ ,\ v\ \rangle$ then $\langle\ S\ ,\ p\ \rangle \Downarrow v$.*

The central meta-theoretic result we'll be interested in is soundness, i.e., every well-typed program terminates and evaluates to a value with the same type as the program.

Definition. *The types of values are given as:*

$$\mathcal{T}(n) = \texttt{i32}$$
$$\mathcal{T}(\texttt{()}) = \texttt{()}$$

Definition. *For any context Γ and store S, we write $\Gamma \sim S$ to mean that $\mathsf{dom}(\Gamma) = \mathsf{dom}(S)$ and $\mathcal{T}(S(x)) = \Gamma(x)$ for all variables $x \in \mathsf{dom}(\Gamma)$.*

Theorem. *(Type Soundness) Let Γ is a context and let S be a store such that $\Gamma \sim S$. For any program p and type t, if $\Gamma \vdash p : t$ then there is a value v such that $\langle\ S\ ,\ p\ \rangle \Downarrow v$ and $\mathcal{T}(v) = t$.*

Proof. We first prove soundness for expressions by induction on derivations.

- (int) Given $\Gamma \vdash n : \texttt{i32}$ take v to be n.
- (unit) Given $\Gamma \vdash \texttt{()} : \texttt{()}$ take v to be $\texttt{()}$.
- (var) Given $\Gamma \vdash x : t$ with $(x \mapsto t) \in \Gamma$, take v to be $S(x)$.
- (add) Suppose $\Gamma \vdash e_1\ \texttt{+}\ e_2 : \texttt{i32}$ where $\Gamma \vdash e_1 : \texttt{i32}$ and $\Gamma \vdash e_2 : \texttt{i32}$. By the IH, there are values v_1 and v_2 such that $\langle\ S\ ,\ e_1\ \rangle \Downarrow v_1$ and $\langle\ S\ ,\ e_2\ \rangle \Downarrow v_2$. By the (add) evaluation rule, we have that $\langle\ S\ ,\ e_1\ \texttt{+}\ e_2\ \rangle \Downarrow v_1 + v_2$, where $\mathcal{T}(v_1 + v_2) = \texttt{i32}$.

We then prove soundness for programs by simultaneous induction on derivations and number of statements in the program.

- If p has no statements, then it is a single expression, and soundness follows from soundness for expressions.

- Otherwise, suppose that $\Gamma_1 \vdash s \texttt{ ; } p : t$ where $\Gamma_1 \vdash s \dashv \Gamma_2$ and $\Gamma_2 \vdash p : t$. Since there is only one form of statement, we know that s is of the form $\texttt{let } x \texttt{ = } e$ and Γ_2 is of the form $\Gamma_1[x \mapsto t]$. Furthermore, there is a type t' such that $\Gamma \vdash e : t'$ and a value v' such that $\langle\ S\ ,\ e\ \rangle \Downarrow v'$ and $\mathcal{T}(v') = t'$ (by soundness for expressions). It suffices to note that $\Gamma[x \mapsto t'] \sim S[x \mapsto v']$ so that we can apply the IH to $\Gamma_1[x \mapsto t'] \vdash p : t$ and get a value v such that $\langle\ S[x \mapsto v']\ ,\ p\ \rangle \Downarrow v$ and $\mathcal{T}(v) = t$. Note that $x \notin \mathsf{dom}(\Gamma)$ by the (let) typing rule, so the $\Gamma[x \mapsto t'] \sim S[x \mapsto v]$ follows from $\Gamma \sim S$ and $\mathcal{T}(S[x \mapsto v](x)) = \mathcal{T}(v) = t' = \Gamma[x \mapsto t'](x)$. We can then use the (let) and (prog) evaluation rules to derive that $\langle\ S\ ,\ s\ \texttt{;}\ p\ \rangle \Downarrow v$.

□