# `salt1`: Mutable Variables, Immutable References

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

## 1 Syntax

The following is the BNF specification for the syntax of `salt1`. We fix an arbitrary set $\mathcal{V}$ of variables.

$$
\begin{array}{rr}
x & \text{(variables, } \mathcal{V}\text{)} \\
n & \text{(integers, } \mathbb{Z}\text{)} \\
\ell ::= \ell_x & \text{(locations, } \mathcal{L}\text{)} \\
w ::= \{\texttt{*}\}\ x & \text{(place expression, } \mathcal{W}\text{)} \\
v ::= \texttt{()} \mid n \mid \ell & \text{(values, Val)} \\
e ::= \texttt{()} \mid n \mid w \mid \texttt{\&}\ w \mid x\ \texttt{=}\ e & \text{(expressions, } \mathcal{E}\text{)} \\
p ::= \{s\ \texttt{;}\}\ e & \text{(programs, } \mathcal{P}\text{)} \\
s ::= e \mid \texttt{let}\ [\texttt{mut}]\ x\ \texttt{=}\ e & \text{(statements, } \mathcal{S}\text{)} \\
t ::= \texttt{i32} \mid \texttt{()} \mid \texttt{\&}\ w & \text{(types, } \mathcal{T}\text{)} \\
m ::= \mathsf{imm} \mid \mathsf{mut} & \text{(mutability)}
\end{array}
$$

## 2 Typing

There are three kinds of typing judgments:

$$
\begin{array}{rr}
\Gamma \vdash e : t \dashv \Gamma & \text{(expressions)} \\
\Gamma \vdash s \dashv \Gamma & \text{(statments)} \\
\Gamma \vdash p : t \dashv \Gamma & \text{(programs)}
\end{array}
$$

$\Gamma$ is a typing context which we will take to be a map from variables to types tagged with mutability information (i.e., a map of the form $\mathcal{V} \mapsto \mathcal{T} \times \{\mathsf{imm}, \mathsf{mut}\}$). Note that expression can have side-effects because of the assignment operator. The following are the typing rules for `salt1`.

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

$$\frac{\Gamma \vdash w_1 : \texttt{\&}\ w_2 \dashv \Gamma \qquad \Gamma \vdash w_2 : t \dashv \Gamma}{\Gamma \vdash \texttt{*}\ w_1 : t \dashv \Gamma}\ \text{(deref)} \qquad \frac{\Gamma \vdash w : t \dashv \Gamma}{\Gamma \vdash \texttt{\&}\ w : \texttt{\&}\ w \dashv \Gamma}\ \text{(imm-borrow)}$$

$$\frac{}{\Gamma \vdash \texttt{i32} \approx \texttt{i32}}\ (\approx\text{-int}) \qquad \frac{}{\Gamma \vdash \texttt{()} \approx \texttt{()}}\ (\approx\text{-unit})$$

$$\frac{\Gamma \vdash w_1 : t_1 \dashv \Gamma \qquad \Gamma \vdash w_2 : t_2 \dashv \Gamma \qquad \Gamma \vdash t_1 \approx t_2}{\Gamma \vdash \texttt{\&}\ w_1 \approx \texttt{\&}\ w_2}\ (\approx\text{-borrow})$$

**Definition.** $\mathsf{writable}(\Gamma, x)$ *is equivalent to* $\nexists y.(y \mapsto \texttt{\&}\ x) \in \Gamma$.

$$\frac{(x \mapsto t_1^{\mathsf{mut}}) \in \Gamma_1 \qquad \Gamma_1 \vdash e : t_2 \dashv \Gamma_2 \qquad \Gamma_2 \vdash t_1 \approx t_2 \qquad \mathsf{writable}(\Gamma_2, x)}{\Gamma_1 \vdash x \texttt{ = } e : \texttt{()} \dashv \Gamma_2[x \mapsto t_2^{\mathsf{mut}}]}\ \text{(assign)}$$

$$\frac{\Gamma_1 \vdash e : t \dashv \Gamma_2}{\Gamma_1 \vdash e \dashv \Gamma_2}\ \text{(expr-stmt)}$$

$$\frac{\Gamma_1 \vdash e : t \dashv \Gamma_2 \qquad x \notin \mathsf{dom}(\Gamma_2)}{\Gamma_1 \vdash \texttt{let}\ x \texttt{ = } e \dashv \Gamma_2[x \mapsto t^{\mathsf{imm}}]}\ \text{(let)} \qquad \frac{\Gamma_1 \vdash e : t \dashv \Gamma_2 \qquad x \notin \mathsf{dom}(\Gamma_2)}{\Gamma_1 \vdash \texttt{let mut}\ x \texttt{ = } e \dashv \Gamma_2[x \mapsto t^{\mathsf{mut}}]}\ \text{(let-mut)}$$

$$\frac{\Gamma_1 \vdash s \dashv \Gamma_2 \qquad \Gamma_2 \vdash p : t \dashv \Gamma_3}{\Gamma_1 \vdash s \texttt{ ; } p : t \dashv \Gamma_3}\ \text{(prog)}$$

# 3 Evaluation

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

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

$S$ stands is a store (a.k.a., dynamic environment) which we will take to be a map from locations to values (i.e., a map of the form $\mathcal{L} \mapsto \mathsf{Val}$). Note that expressions can have side-effects because of the assignment operator. The following are the evaluation rules for `salt1`.

$$\frac{}{\langle\ S\ ,\ \texttt{()}\ \rangle \Downarrow \langle\ S\ ,\ \texttt{()}\ \rangle}\ \text{(unit)} \qquad \frac{}{\langle\ S\ ,\ n\ \rangle \Downarrow \langle\ S\ ,\ n\ \rangle}\ \text{(int)}$$

**Definition.** *The functions* $\mathsf{loc} : (\mathcal{L} \mapsto \mathsf{Val}) \times \mathcal{W} \to \mathcal{L}$ *and* $\mathsf{read} : (\mathcal{L} \mapsto \mathsf{Val}) \times \mathcal{W} \to \mathsf{Val}$ *are given as:*

$$\begin{aligned} \mathsf{loc}(S, x) &= \ell_x \\ \mathsf{loc}(S, \texttt{*}\ w) &= S(\mathsf{loc}(S, w)) \\ \mathsf{read}(S, w) &= S(\mathsf{loc}(S, w)) \end{aligned}$$

$$\frac{}{\langle\ S\ ,\ w\ \rangle \Downarrow \langle\ S\ ,\ \mathsf{read}(S, w)\ \rangle}\ \text{(place)} \qquad \frac{}{\langle\ S\ ,\ \texttt{\&}\ w\ \rangle \Downarrow \langle\ S\ ,\ \mathsf{loc}(S, w)\ \rangle}\ \text{(imm-borrow)}$$

$$\frac{\langle\ S_1\ ,\ e\ \rangle \Downarrow \langle\ S_2\ ,\ v\ \rangle}{\langle\ S_1\ ,\ x \texttt{ = } e\ \rangle \Downarrow\ \langle\ S_2[\ell_x \mapsto v]\ ,\ \texttt{()}\ \rangle}\ \text{(assign)}$$

$$\frac{\langle\ S_1\ ,\ e\ \rangle \Downarrow \langle\ S_2\ ,\ v\ \rangle}{\langle\ S_1\ ,\ e\ \rangle \Downarrow S_2}\ \text{(expr-stmt)} \qquad \frac{\langle\ S_1\ ,\ e\ \rangle \Downarrow \langle\ S_2\ ,\ v\ \rangle}{\langle\ S_1\ ,\ \texttt{let}\ [\texttt{mut}]\ x \texttt{ = } e\ \rangle \Downarrow S_2[\ell_x \mapsto v]}\ \text{(let)}$$

$$\frac{\langle\ S_1\ ,\ s\ \rangle \Downarrow S_2 \qquad \langle\ S_2\ ,\ p\ \rangle \Downarrow \langle\ S_3\ ,\ v\ \rangle}{\langle\ S_1\ ,\ s \texttt{ ; } p\ \rangle \Downarrow \langle\ S_3\ ,\ v\ \rangle}\ \text{(prog)}$$

# 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 deal to included holes:

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

We don't need a dummy statement $\bullet$ because expressions are considered expressions The following are the reduction rules for `salt1`.

$$\frac{}{\langle\, S\,,\, w\,\rangle \longrightarrow \langle\, S\,,\, \mathsf{read}(S, w)\,\rangle}\ \text{(place)}$$

$$\frac{}{\langle\, S_1\,,\, \&\, w\,\rangle \longrightarrow \langle\, S\,,\, \mathsf{loc}(S, w)\,\rangle}\ \text{(imm-borrow)}$$

$$\frac{}{\langle\, S\,,\, x = v\,\rangle \longrightarrow \langle\, S[\ell_x \mapsto v]\,,\, ()\,\rangle}\ \text{(assign)}$$

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

$$\frac{\langle\, S\,,\, s_1\,\rangle \longrightarrow \langle\, S'\,,\, s_1'\,\rangle}{\langle\, S\,,\ s_1\ ;\ldots;\ s_k\ ;\ e\,\rangle \longrightarrow \langle\, S'\,,\, s_1'\ ;\ldots;\ s_k\ ;\ e\,\rangle}\ \text{(prog}_1\text{)}$$

$$\frac{}{\langle\, S\,,\, v\ ;\ s_2\ ;\ldots;\ s_k\ ;\ e\,\rangle \longrightarrow \langle\, S\,,\, s_2\ ;\ldots;\ s_k\ ;\ e\,\rangle}\ \text{(prog}_2\text{)}$$

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

# 5 Meta-Theory

**Theorem.** *(Operational Adequacy) For any stores $S_1$, program $p$, and value $v$, if $\langle\, S_1\,,\, p\,\rangle \Downarrow v$, then there is a store $S_2$ such that $\langle\, S_1\,,\, p\,\rangle \longrightarrow^\star \langle\, S_2\,,\, v\,\rangle$.*

**Theorem.** *(Operational Soundness) For any stores $S_1$ and $S_2$, program $p$, and value $v$, if $\langle\, S_1\,,\, p\,\rangle \longrightarrow^\star \langle\, S_2\,,\, v\,\rangle$ then $\langle\, S_1\,,\, p\,\rangle \Downarrow v$.*

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

$$\mathcal{T}(n) = \texttt{i32}$$
$$\mathcal{T}(()) = ()$$
$$\mathcal{T}(\ell_x) = \&\, x$$

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

**Theorem.** *(Type Soundness) Let $\Gamma$ is a context, let $S_1$ be a store such that $\Gamma_1 \sim S_1$. For any program $p$ and type $t$, if $\Gamma_1 \vdash p : t \dashv \Gamma_2$ then there is a store $S_2$ value $v$ such that $\langle\ S_1\ ,\ p\ \rangle \Downarrow \langle\ S_2\ ,\ v\ \rangle$ and $\Gamma_2 \sim S_2$ and $\Gamma_2 \vdash t \approx \mathcal{T}(v)$.*

*Proof.* We first prove soundness for expression by induction on derivations.

- (unit) Obvious.
- (int) Obvious.
- (var) This follow immediately from the (var) rules and $\Gamma_1 \sim S_1$.
- (deref) By the IH, we know that $\langle\ S_1\ ,\ w_1\ \rangle \Downarrow \langle\ S_1\ ,\ \mathsf{read}(S_1, w_1)\ \rangle$ where $\Gamma_1 \vdash \texttt{\&}\ w_2 \approx \mathcal{T}(\mathsf{read}(S_1, w_1))$ and $\langle\ S_1\ ,\ w_2\ \rangle \Downarrow \langle\ S_1\ ,\ \mathsf{read}(S_1, w_2)\ \rangle$ where $\Gamma_1 \vdash t \approx \mathcal{T}(\mathsf{read}(S_1, w_2))$. In order for the former type equivalence to be derivable, it must be that $\mathsf{read}(S_1, w_1) = S_1(\mathsf{loc}(S_1, w_1)) = \ell_z$ for some variable $z$ so that $\mathcal{T}(S_1(\mathsf{loc}(S_1, w_1))) = \texttt{\&}\ z$. Furthermore, this equivalence must follow from the ($\approx$-borrow) rule, which implies that $\Gamma_1 \vdash t \approx \Gamma_1(z)$ (exercise). Next, note that

  $$\mathcal{T}(\mathsf{read}(S_1, \texttt{*}\ w_1)) = \mathcal{T}(S_1(S_1(\mathsf{loc}(S_1, w_1)))) = \mathcal{T}(S_1(\ell_z))$$

  and $\Gamma_1 \vdash \Gamma_1(z) \approx \mathcal{T}(S_1(\ell_z))$ since $\Gamma_1 \sim S_1$. Finally, $\Gamma \vdash \cdot \approx \cdot$ defines an equivalence relation on types (exercise), meaning that $\Gamma_1 \vdash t \approx \mathcal{T}(\mathsf{read}(S_1, \texttt{*}\ w_1))$ as desired.
- (imm-borrow) By the IH, We know that $\langle\ S_1\ ,\ w\ \rangle \Downarrow \langle\ S_1\ ,\ S_1(\mathsf{loc}(S_1, w)\ \rangle$ and $\Gamma_1 \vdash t \approx \mathcal{T}(S_1(\mathsf{loc}(S_1, w))$. We can take $v$ to be $\mathsf{loc}(S_1, w)$ by the (imm-borrow) evaluation rule. We need to verify that $\Gamma_1 \vdash \texttt{\&}\ w \approx \mathcal{T}(\mathsf{loc}(S_1, w))$. Since $\mathsf{loc}(S_1, w)$ appears as an argument to $S_1$ in a type equivalence, it be of the form $\ell_z$ for some variable $z$. Thus, it suffices to note that $\Gamma_1 \vdash \Gamma_1(z) \approx \mathcal{T}(S_1(\ell_z))$ by $\Gamma_1 \sim S_1$ and so we can derive:

  $$\frac{\Gamma \vdash w : t \dashv \Gamma \qquad \Gamma \vdash z : \Gamma(z) \dashv \Gamma \qquad \Gamma \vdash t \approx \Gamma(z)}{\Gamma \vdash \texttt{\&}\ w \approx \mathcal{T}(\mathsf{loc}(S, w))}$$

- (assign) By the IH, we have a store $S_2$ and value $v$ such that $\langle\ S_1\ ,\ e\ \rangle \Downarrow \langle\ S_2\ ,\ v\ \rangle$ where $\Gamma_2 \sim S_2$ and $\Gamma_2 \vdash t_2 \approx \mathcal{T}(v)$. By the (assign) evaluation rule, we have $\langle\ S_1\ ,\ x\ \texttt{=}\ e\ \rangle \Downarrow \langle\ S_2[\ell_x \mapsto v]\ ,\ \texttt{()}\ \rangle$. We need to show that $\Gamma_2[x \mapsto t_2] \sim S_2[\ell_x \mapsto v]$. This follows from the fact that $\Gamma_2 \vdash t_1 \approx t_2$ by assumption and that type equivalence is an equivalence relation ($x$ is the only variable we need to check for the $\sim$-condition).

Next we prove soundness for statements by induction on derivations.

- (expr-stmt) This follows directly from soundness for expressions
- (let-[mut]) By the IH, there is a store $S_2$ and a value $v$ such that $\langle\ S_1\ ,\ e\ \rangle \Downarrow \langle\ S_2\ ,\ v\ \rangle$ and $\Gamma_2 \sim S_2$ and $\Gamma_2 \vdash t \approx \mathcal{T}(v)$. By the (let) evaluation rule $\langle\ S_1\ ,\ \texttt{let}\ [\texttt{mut}]\ x\ \texttt{=}\ e\ \rangle \Downarrow S_2[\ell_x \mapsto v]$. It follows immediately that $\Gamma_2[x \mapsto t^m] \sim S_2[\ell_x \mapsto v]$; in particular $\Gamma_2[x \mapsto t^m] \vdash \Gamma(x) \approx \mathcal{T}(S_2[\ell_x \mapsto v](\ell_x))$.

Soundness of programs then follows immediately by induction on derivations using the (prog) rule. □