Opal Semantics

Alex Renda

November 8, 2017

1 Description

Tiny turing-incomplete language based on Imp with sexps:

2 Grammar

$\langle Com \rangle$ ::= `skip`
  | $\langle Com \rangle$; $\langle Com \rangle$
  | `if` $\langle Bool \rangle$ `then` $\{\langle Com \rangle\}$ `else` $\{\langle Com \rangle\}$
  | `with` $\langle Node \rangle$ `do` $\{\langle Com \rangle\}$
  | `at` $\langle Node \rangle$ `do` $\{\langle Com \rangle\}$
  | $\langle World \rangle$:= `hyp` $\{\langle Com \rangle\}$
  | `commit` $\langle World \rangle$
  | `handle` $\langle Node \rangle.\langle Var \rangle$:= $\langle Op \rangle$ `with` $\langle Sexp \rangle$ `merging` $\langle Var \rangle$ $\langle Var \rangle$ $\langle Var \rangle$ `to` $\langle Sexp \rangle$ `in` $\{\langle Com \rangle\}$
  | $\langle Op \rangle$

$\langle Sexp \rangle$ ::= $\varnothing$
  | $\langle Node \rangle.\langle Var \rangle$
  | ($\langle Sexp \rangle$. $\langle Sexp \rangle$)

$\langle Bool \rangle$ ::= $\langle Sexp \rangle = \langle Sexp \rangle$
  | $\langle Sexp \rangle \in \langle Sexp \rangle$
  | $\langle Bool \rangle \wedge \langle Bool \rangle$
  | $\langle Bool \rangle \vee \langle Bool \rangle$

$|$ `true`
$|$ `false`

2.1 Values

$\langle \mathit{SexpValue} \rangle$ ::= $\varnothing$
$|$ ($\langle \mathit{SexpValue} \rangle$. $\langle \mathit{SexpValue} \rangle$)

$\langle \mathit{BoolValue} \rangle$ ::= `true`
$|$ `false`

3 Typing Judgement

Well-formedness conditions:

- all used operations are defined
- all node/var accesses are defined
- all node accesses are permissioned
- all committed worlds are defined
- worlds are committed at most once (affine types)

3.1 Terminology

Σ :	$2^{\langle \mathit{Node} \rangle \times \langle \mathit{Var} \rangle}$	Set of variables in scope
H :	$2^{\langle \mathit{Op} \rangle}$	Set of handlers in scope
Π :	$2^{\langle \mathit{Node} \rangle}$	Set of nodes giving permission
Ω :	$2^{\langle \mathit{World} \rangle}$	Set of worlds in scope

3.2 Commands

$$\frac{}{\Omega, \Pi, \Sigma, H \vdash \texttt{skip} : \Omega}$$

$$\frac{\Omega, \Pi, \Sigma, H \vdash c_1 : \Omega' \qquad \Omega', \Pi, \Sigma, H \vdash c_2 : \Omega''}{\Omega, \Pi, \Sigma, H \vdash c_1; c_2 : \Omega''}$$

$$\frac{\Sigma \vdash b : \textsc{Bool} \qquad \Omega', \Pi, \Sigma, H \vdash c_1 : \Omega' \qquad \Omega', \Pi, \Sigma, H \vdash c_2 : \Omega''}{\Omega, \Pi, \Sigma, H \vdash \texttt{if}\ b\ \texttt{then}\ \{c_1\}\ \texttt{else}\ \{c_2\} : \Omega' \cap \Omega''}$$

$$\frac{\varnothing, \Pi, \Sigma, H \vdash c : \Omega'}{\Omega, \Pi, \Sigma, H \vdash u\ \texttt{:=}\ \texttt{hyp}\ \{c\} : \Omega \cup \{u\}}$$

$$\frac{u \in \Omega}{\Omega, \Pi, \Sigma, H \vdash \texttt{commit}\ u : \Omega \setminus \{u\}}$$

$$\frac{n \in \Pi \qquad \Sigma \cup \{(n, v)\} \vdash s_h : \textsc{Sexp} \qquad \Sigma \cup \{(n, v_o), (n, v_h), (n, v_c)\} \vdash s_m : \textsc{Sexp} \qquad \varnothing, \Pi, \Sigma \cup \{(n, v)\}, H \cup \{op\} \vdash c : \Omega'}{\Omega, \Pi, \Sigma, H \vdash \texttt{handle}\ n.v\ \texttt{:=}\ op\ \texttt{with}\ s_h\ \texttt{merging}\ v_o\ v_h\ v_c\ \texttt{to}\ s_m\ \texttt{in}\ \{c\} : \Omega}$$

$$\frac{op \in H}{\Omega, \Pi, \Sigma, H \vdash op : \Omega}$$

$$\frac{\Omega, \Pi \cup \{n\}, \Sigma, H \vdash c : \Omega'}{\Omega, \Pi, \Sigma, H \vdash \texttt{with}\ n\ \texttt{do}\ \{c\} : \Omega'}$$

$$\frac{\Omega, \Pi, \Sigma, H \vdash c : \Omega'}{\Omega, \Pi, \Sigma, H \vdash \texttt{at}\ n\ \texttt{do}\ \{c\} : \Omega'}$$

3.3 Booleans

$$\frac{}{\Sigma \vdash \texttt{true} : \textsc{Bool}}$$

$$\frac{}{\Sigma \vdash \texttt{false} : \textsc{Bool}}$$

$$\frac{\Sigma \vdash b_1 : \textsc{Bool} \qquad \Sigma \vdash b_2 : \textsc{Bool}}{\Sigma \vdash b_1 \wedge b_2 : \textsc{Bool}}$$

$$\frac{\Sigma \vdash b_1 : \textsc{Bool} \qquad \Sigma \vdash b_2 : \textsc{Bool}}{\Sigma \vdash b_1 \vee b_2 : \textsc{Bool}}$$

$$\frac{\Sigma \vdash s_1 : \textsc{Sexp} \qquad \Sigma \vdash s_2 : \textsc{Sexp}}{\Sigma \vdash s_1 = s_2 : \textsc{Bool}}$$

$$\frac{\Sigma \vdash s_1 : \textsc{Sexp} \qquad \Sigma \vdash s_2 : \textsc{Sexp}}{\Sigma \vdash s_1 \in s_2 : \textsc{Bool}}$$

3.4 Sexps

$$\frac{}{\Sigma \vdash \varnothing : \textsc{Sexp}}$$

$$\frac{(n, v) \in \Sigma}{\Sigma \vdash n.v : \textsc{Sexp}}$$

$$\frac{\Sigma \vdash s_1 : \textsc{Sexp} \qquad \Sigma \vdash s_2 : \textsc{Sexp}}{\Sigma \vdash (s_1.s_2) : \textsc{Sexp}}$$

4 Small Step Semantics

4.1 Terminology

σ :	$\langle Node\rangle \times \langle Var\rangle \rightharpoonup \langle SexpValue\rangle$	function representing each node's store
Σ :	$\bullet \mid (\sigma, \Sigma)$	stack of stores (for Hyp and At)
ω :	$\langle World\rangle \rightharpoonup (\Sigma \times \sigma)$	function representing each world's initial stack and final store
π :	$2^{\langle Node\rangle}$	current authorized set of principals
ρ :	$\langle Node\rangle$	current execution location
η :	$\langle Op\rangle \rightharpoonup (\langle Node\rangle \times \langle Var\rangle \times \langle Sexp\rangle)$	mapping of handlers to their destinations and expressions
μ :	$(\langle Node\rangle \times \langle Var\rangle) \rightharpoonup (\langle Var\rangle \times \langle Var\rangle \times \langle Var\rangle \times \langle Sexp\rangle)$	mapping of handler results to their merge expressions
$\Downarrow_{\langle Com\rangle}$:	$((\langle Com\rangle \times \Sigma \times \omega \times \pi \times \rho \times \eta \times \mu) \rightharpoonup (\sigma \times \omega)$	Commands step to a new top-level store and new set of hypothetical worlds
$\Downarrow_{\langle Sexp\rangle}$:	$((\langle Sexp\rangle \times \Sigma \times \pi) \rightharpoonup \langle SexpValue\rangle$	Sexps step to a value, or nothing if it cannot be evaluated due to undefined variables or lack of permission
$\Downarrow_{\langle Bool\rangle}$:	$((\langle Bool\rangle \times \Sigma \times \pi) \rightharpoonup \langle BoolValue\rangle$	Bools step to a value, or nothing if it cannot be evaluated due to undefined variables or lack of permission

4.2 Basic commands

$$\frac{}{\langle \texttt{skip}, (\sigma, \Sigma), \omega, \pi, \rho, \eta, \mu\rangle \Downarrow \langle \sigma, \omega\rangle}\ \textsc{Skip}$$

$$\frac{\langle c_1, (\sigma, \Sigma), \omega, \pi, \rho, \eta, \mu \rangle \Downarrow \langle \sigma', \omega' \rangle \qquad \langle c_2, (\sigma', \Sigma), \omega', \pi, \rho, \eta, \mu \rangle \Downarrow \langle \sigma'', \omega'' \rangle}{\langle c_1;\ c_2, (\sigma, \Sigma), \omega, \pi, \rho, \eta, \mu \rangle \Downarrow \langle \sigma'', \omega'' \rangle} \textsc{Seq}$$

$$\frac{\langle b, \Sigma, \pi \rangle \Downarrow \texttt{true} \qquad \langle c_1, \Sigma, \omega, \pi, \rho, \eta, \mu \rangle \Downarrow \langle \sigma', \omega' \rangle}{\langle \texttt{if } b \texttt{ then } \{c_1\} \texttt{ else } \{c_2\}, \Sigma, \omega, \pi, \rho, \eta, \mu \rangle \Downarrow \langle \sigma', \omega' \rangle} \textsc{If-True}$$

$$\frac{\langle b, \Sigma, \pi \rangle \Downarrow \texttt{false} \qquad \langle c_2, \Sigma, \omega, \pi, \rho, \eta, \mu \rangle \Downarrow \langle \sigma', \omega' \rangle}{\langle \texttt{if } b \texttt{ then } \{c_1\} \texttt{ else } \{c_2\}, \Sigma, \omega, \pi, \rho, \eta, \mu \rangle \Downarrow \langle \sigma', \omega' \rangle} \textsc{If-False}$$

4.3 Distribution commands

In this semantics, At is effectively a noop/passthrough, resulting in behavior that would be identical with or without it (modulo endorsement taking location into account). It does have real-world semantics though: At represents executing a command on another computer, starting with a new empty store on the top of the stack, and shipping back the new store at the end. Beyond this, exact semantics are implementation dependent. An implementation could, in theory, send the entire stack of stores over in addition to just the command. Alternatively, the new machine could request any specific variables it needs, to mitigate the amount of data shuttled over the network.

With is also a passthrough in a similar regard: its runtime effects consist of asking the specified user for permission to run the specified command on the current machine (represented by $\langle n, \rho, c \rangle \checkmark$ in the semantics – the details of this function are implementation dependent).

$$\frac{\langle c, (\sigma_\varnothing, (\sigma, \Sigma)), \omega, \pi, n, \eta, \mu \rangle \Downarrow \langle \sigma', \omega' \rangle}{\langle \texttt{at } n \texttt{ do } \{c\}, (\sigma, \Sigma), \omega, \pi, \rho, \eta, \mu \rangle \Downarrow \langle \sigma'; \sigma, \omega' \rangle} \textsc{At}$$

$$\frac{\langle n, \rho, c \rangle \checkmark \qquad \langle c, \Sigma, \omega, \pi \cup \{n\}, \rho, \eta, \mu \rangle \Downarrow \langle \sigma', \omega' \rangle}{\langle \texttt{with } n \texttt{ do } \{c\}, \Sigma, \omega, \pi, \rho, \eta, \mu \rangle \Downarrow \langle \sigma', \omega' \rangle} \textsc{With}$$

4.4 Handler commands

We describe handlers as a tuple of $(\langle \mathit{Node} \rangle, \langle \mathit{Var} \rangle, \langle \mathit{Op} \rangle, \langle \mathit{Sexp} \rangle_h, \langle \mathit{Sexp} \rangle_m, \langle \mathit{Com} \rangle)$, where $\langle \mathit{Node} \rangle$ is the node to write results to, $\langle \mathit{Var} \rangle$ is the variable on that node to write results to, $\langle \mathit{Op} \rangle$ is the name of the handler, $\langle \mathit{Sexp} \rangle_h$ is a sexp that gets lazily evaluated with its results written to $\langle \mathit{Node} \rangle.\langle \mathit{Var} \rangle$ each time $\langle \mathit{Op} \rangle$ is called, and $\langle \mathit{Sexp} \rangle_m$ is a sexp that gets lazily evaluated upon merge conflicts within a variable upon `commit`ing a hypothetical world, with $\varnothing$.Orig set to the original value of the variable before the hypothetical execution, $\varnothing$.Hyp set to the value of the variable after the hypothetical execution, and $\varnothing$.Curr set to the value of the variable in the context of where `commit` is being called. Immediately within the scope of a handler, the handled variable is set to $\varnothing$.

$$\frac{\langle c, (\sigma[(n,v) \mapsto \varnothing], \Sigma), \omega, \pi, \rho, \eta[op \mapsto (n, v, s_h)], \mu[(n,v) \mapsto (v_i, v_h, v_c, s_m)] \rangle \Downarrow \langle \sigma', \omega' \rangle \qquad n \in \pi}{\langle \texttt{handle}\ n.v\ := op\ \texttt{with}\ s_h\ \texttt{merging}\ v_o\ v_h\ v_c\ \texttt{to}\ s_m\ \texttt{in}\ \{c\}, (\sigma, \Sigma), \omega, \pi, \rho, \eta, \mu \rangle \Downarrow \langle \sigma', \omega' \rangle} \textsc{Handle}$$

$$\frac{\eta(op) = (n, v, s_h) \qquad \langle s_h, (\sigma, \Sigma), \pi \rangle \Downarrow s}{\langle op, (\sigma, \Sigma), \omega, \pi, \rho, \eta, \mu \rangle \Downarrow \langle \sigma[(n,v) \mapsto s], \omega \rangle} \textsc{Op}$$

4.5 Hypothetical commands

$$\frac{\langle c, (\sigma_\varnothing, (\sigma, \Sigma)), \omega_\varnothing, \pi, \rho, \eta, \mu \rangle \Downarrow \langle \sigma', \omega' \rangle}{\langle u\ :=\ \texttt{hyp}\ \{c\}, (\sigma, \Sigma), \omega, \pi, \rho, \eta, \mu \rangle \Downarrow \langle \sigma, \omega[u \mapsto ((\sigma, \Sigma), \sigma')] \rangle} \textsc{Hyp}$$

$$\frac{\langle n.v, \Sigma_{orig}, \pi \rangle \Downarrow s_o \qquad \langle n.v, (\sigma_{hyp}, \Sigma_{orig}), \pi \rangle \Downarrow s_h \qquad \langle n.v, (\sigma_{curr}, \Sigma_{curr}), \pi \rangle \Downarrow s_c}{\sigma_{merge} = [(n, vo) \mapsto s_o, (n, vh) \mapsto s_h, (n, vc) \mapsto s_c]} \textsc{MergeStore}$$

$$\frac{\mu((n,v)) = s_m v_o, v_h, v_c \qquad \textsc{MergeStore} \qquad \langle s_m, (\sigma_{merge}, (\sigma_{curr}, \Sigma_{curr})), \pi \rangle \Downarrow v_m}{\sigma'[(n,v) \mapsto v_m]} \textsc{MergesTo}$$

$$\frac{\omega(u) = (\Sigma_{\texttt{orig}}, \sigma_{\texttt{hyp}}) \qquad \forall (n,v,s) \in \sigma_{\texttt{hyp}}, \textsc{MergesTo} \qquad \forall (n,v), \neg\exists s.(n,v,s) \in \sigma_{\texttt{hyp}} \Rightarrow \neg\exists s.(n,v,s) \in \sigma'}{\langle \texttt{commit}\ u, (\sigma_{curr}, \Sigma_{curr}), \omega, \pi, \rho, \eta, \mu \rangle \Downarrow \langle \sigma', \omega \rangle} \textsc{Commit}$$

4.6 Sexps

$$\frac{}{\langle \varnothing, \Sigma, \pi \rangle \Downarrow \varnothing} \textsc{EmptySet}$$

$$\frac{\langle s_1, \Sigma, \pi \rangle \Downarrow s_{v1} \qquad \langle s_2, \Sigma, \pi \rangle \Downarrow s_{v2}}{\langle (s_1.s_2), \Sigma, \pi \rangle \Downarrow (s_{v1}.s_{v2})} \textsc{Cons}$$

$$\frac{\Sigma(n,v) = s_v}{\langle n.v, \Sigma, \pi \rangle \Downarrow s_v} \textsc{Var}$$

4.7 Bools

$$\frac{}{\langle \texttt{true}, \Sigma, \pi\rangle \Downarrow \texttt{true}}\ \textsc{True}$$

$$\frac{}{\langle \texttt{false}, \Sigma, \pi\rangle \Downarrow \texttt{false}}\ \textsc{False}$$

w $$\frac{\langle b_1, \Sigma, \pi\rangle \Downarrow \texttt{true} \qquad \langle b_2, \Sigma, \pi\rangle \Downarrow \texttt{true}}{\langle b_1 \wedge b_2, \Sigma, \pi\rangle \Downarrow \texttt{true}}\ \textsc{AndTrue}$$

$$\frac{\langle b_1, \Sigma, \pi\rangle \Downarrow \texttt{false}}{\langle b_1 \wedge b_2, \Sigma, \pi\rangle \Downarrow \texttt{false}}\ \textsc{AndFalseL}$$

$$\frac{\langle b_2, \Sigma, \pi\rangle \Downarrow \texttt{false}}{\langle b_1 \wedge b_2, \Sigma, \pi\rangle \Downarrow \texttt{false}}\ \textsc{AndFalseR}$$

$$\frac{\langle b_1, \Sigma, \pi\rangle \Downarrow \texttt{false} \qquad \langle b_2, \Sigma, \pi\rangle \Downarrow \texttt{false}}{\langle b_1 \vee b_2, \Sigma, \pi\rangle \Downarrow \texttt{false}}\ \textsc{OrFalse}$$

$$\frac{\langle b_1, \Sigma, \pi\rangle \Downarrow \texttt{true}}{\langle b_1 \vee b_2, \Sigma, \pi\rangle \Downarrow \texttt{true}}\ \textsc{OrTrueL}$$

$$\frac{\langle b_2, \Sigma, \pi\rangle \Downarrow \texttt{true}}{\langle b_1 \vee b_2, \Sigma, \pi\rangle \Downarrow \texttt{true}}\ \textsc{OrTrueR}$$

$$\frac{\langle s_1, \Sigma, \pi\rangle \Downarrow \varnothing \qquad \langle s_2, \Sigma, \pi\rangle \Downarrow \varnothing}{\langle s_1 = s_2, \Sigma, \pi\rangle \Downarrow \texttt{true}}\ \textsc{EqTrue}$$

$$\frac{\langle s_1, \Sigma, \pi\rangle \Downarrow (s_{v11}.s_{v12}) \qquad \langle s_2, \Sigma, \pi\rangle \Downarrow \varnothing}{\langle s_1 = s_2, \Sigma, \pi\rangle \Downarrow \texttt{false}}\ \textsc{EqFalseL}$$

$$\frac{\langle s_1, \Sigma, \pi\rangle \Downarrow \varnothing \qquad \langle s_2, \Sigma, \pi\rangle \Downarrow (s_{v21}.s_{v22})}{\langle s_1 = s_2, \Sigma, \pi\rangle \Downarrow \texttt{false}}\ \textsc{EqFalseR}$$

$$\frac{\langle s_1, \Sigma, \pi\rangle \Downarrow (s_{v11}.s_{v12}) \qquad \langle s_2, \Sigma, \pi\rangle \Downarrow (s_{v21}.s_{v22}) \qquad \langle s_{v11} = s_{v21} \wedge s_{v12} = s_{v22}, \Sigma, \pi\rangle \Downarrow b}{\langle s_1 = s_2, \Sigma, \pi\rangle \Downarrow b}\ \textsc{EqProp}$$

$$\frac{\langle s_1, \Sigma, \pi\rangle \Downarrow s_{v1} \qquad \langle s_2, \Sigma, \pi\rangle \Downarrow \varnothing}{\langle s_1 \in s_2, \Sigma, \pi\rangle \Downarrow \texttt{false}}\ \textsc{MemFalse}$$

$$\frac{\langle s_1, \Sigma, \pi\rangle \Downarrow s_{v1} \qquad \langle s_2, \Sigma, \pi\rangle \Downarrow (s_{v21}.s_{v22}) \qquad \langle s_{v1} = s_{v21} \vee s_{v1} \in s_{v22}, \Sigma, \pi\rangle \Downarrow b}{\langle s_1 \in s_2, \Sigma, \pi\rangle \Downarrow b}\ \textsc{MemProp}$$

EqProp and MemProp are well-founded proof trees since the sexp step is idempotent, and the two props decrease on size of sexp, so the maximum depth of the proof tree is proportional to the maximum depth of the sexp values