The Semantics of CML (From Appendix B of Reppy's Book)

In these slides, we present the syntax and most of the dynamic semantics of a subset of CML.

There are some bugs in what follows.

B.1: Notation

- We write $A \setminus B$ for $\{ x \in A \mid x \notin B \}$.
- We write $A \stackrel{\text{fin}}{\to} B$ for the set of all *finite maps* from A to B, i.e., the set of all finite partial functions from A to B.
- If $f \in A \stackrel{\text{fin}}{\to} B$, then

$$\mathrm{dom}(f) = \{ x \mid f(x) \text{ is defined} \},$$
$$\mathrm{rng}(f) = \{ f(x) \mid x \in \mathrm{dom}(f) \}.$$

- If $A' \subset A$, A' is finite and $b \in B$, then $\{A' \mapsto b\} \in A \stackrel{\text{fin}}{\to} B$ is the constantly b map whose domain is A'.
- If f and g are finite maps, then $f \pm g$ is the finite map with domain $\mathrm{dom}(f) \cup \mathrm{dom}(g)$ such that, for all $x \in \mathrm{dom}(f) \cup \mathrm{dom}(g)$,

$$(f \pm g)(x) = \begin{cases} g(x) & \text{if } x \in \mathrm{dom}(g), \\ f(x) & \text{otherwise.} \end{cases}$$

B.2: Syntax of Mini-CML

$x \in \mathrm{Var}$	variables
$b \in \mathrm{Const} = \mathrm{BConst} \cup \mathrm{FConst}$	constants
$\mathrm{BConst} = \{(), \texttt{true}, \texttt{false}, 0, 1, \ldots\}$	base constants
$\mathrm{FConst} = \{+, -, \texttt{fst}, \texttt{snd}, \ldots\}$	function constants

FConst includes `alwaysEvt`, `channel`, `choose`, `guard`, `neverEvt`, `recvEvt`, `sendEvt`, `wrap` and `withNack`.

B.2: Surface Syntax

The surface syntax of Mini-CML is defined by:

$e ::= x$	variables
$\mid \quad b$	constants
$\mid \quad (e_1, e_2)$	pairs
$\mid \quad \mathbf{fn}\ x \Rightarrow e$	function abstraction
$\mid \quad (e_1\ e_2)$	function application
$\mid \quad \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2$	let binding
$\mid \quad \mathbf{spawn}\ e$	process creation
$\mid \quad \mathbf{sync}\ e$	synchronization

B.2: Internal Syntax

To give the internal syntax of Mini-CML, we need:

$\kappa \in \mathrm{Ch}$	channel names
$\gamma \in \mathrm{Cond}$	condition (event) names

B.2: Internal Syntax (Cont.)

We form the internal syntax by extending the surface syntax:

$$
\begin{array}{lll}
e ::= \ldots & & \\
\quad | \quad \kappa & & \text{channel name} \\
\quad | \quad \gamma & & \text{condition event} \\
\quad | \quad e? & & \text{receive event} \\
\quad | \quad e_1 \,!\, e_2 & & \text{send event} \\
\quad | \quad e_1 \Rightarrow e_2 & & \text{wrap event} \\
\quad | \quad \mathbf{N}\, e & & \text{guarded event function with nack} \\
\quad | \quad e_1 \oplus e_2 & & \text{event choice} \\
\quad | \quad e_1 \mid e_2 & & \text{abort wrapper} \\
\quad | \quad \Lambda & & \text{never event}
\end{array}
$$

B.2: Values

$$
\begin{aligned}
v ::= {} & b \\
\mid {} & \kappa \\
\mid {} & (v_1, v_2) \\
\mid {} & \mathbf{fn}\ x \Rightarrow e \\
\mid {} & \mathbf{N}\, v \\
\mid {} & ev
\end{aligned}
$$

B.2: Event Values

$$
\begin{aligned}
ev ::= \; & \kappa? \\
| \quad & \kappa \,!\, v \\
| \quad & \gamma \\
| \quad & ev \Rightarrow v \\
| \quad & ev_1 \oplus ev_2 \\
| \quad & ev \mid \gamma \\
| \quad & \Lambda
\end{aligned}
$$

B.2: Syntactic Classes

$e \in \mathrm{Exp}$	expressions
$v \in \mathrm{Val} \subset \mathrm{Exp}$	values
$ev \in \mathrm{Event} \subset \mathrm{Val}$	event values

B.3: Dynamic Semantics

- $\mathcal{K} \subset \mathrm{Ch}$—finite sets of channels.
- $\mathcal{C} \subset \mathrm{Cond} \xrightarrow{\mathrm{fin}} \{\texttt{true}, \texttt{false}\}$—state of a collection of condition events.
- Each Mini-CML process has a unique process identifier $\pi \in \mathrm{ProcId}$.
- $\mathcal{P} \in \mathrm{ProcId} \xrightarrow{\mathrm{fin}} \mathrm{Exp}$—state of a set of processes.
- Configuration $\mathcal{C}, \mathcal{K}, \mathcal{P}$—state of a Mini-CML computation.

B.3: Four Reduction Relations

- Concurrent Evaluation:

$$\mathcal{C}, \mathcal{K}, \mathcal{P} \Rightarrow \mathcal{C}', \mathcal{K}', \mathcal{P}'.$$

- Rendezvous Projection:

$$\mathcal{C}, \mathcal{P} \mapsto (\mathcal{P}', \mathcal{P}'')/\mathcal{A},$$

  where $\mathcal{P}' \subset \mathcal{P}$, $\mathrm{dom}(\mathcal{P}') = \mathrm{dom}(\mathcal{P}'')$, and $\mathcal{A} \subset \mathrm{Cond}$ is a finite set of negative acknowledgment conditions.

- Event-matching Relations:

$$\mathcal{C} \vdash (ev_1, \ldots, ev_k) \rightsquigarrow_k (e_1, \ldots, e_k)/\mathcal{A}.$$

- Sequential Evaluation:

$$\mathcal{C}, \mathcal{K}, e \hookrightarrow \mathcal{C}', \mathcal{K}', e'.$$

B.3: Evaluation Contexts

Order of evaluation is specified using evaluation contexts:

$$
\begin{array}{rcl}
E & ::= & [\,] \\
& | & (E\,e) \\
& | & (v\,E) \\
& | & (E, e) \\
& | & (v, E) \\
& | & \mathbf{let}\ x = E\ \mathbf{in}\ e \\
& | & \mathbf{spawn}\,E \\
& | & \mathbf{sync}\,E \\
& | & E \mid \gamma
\end{array}
$$

$E[e]$ means replace $[\,]$ in E by e.

B.3.1: Concurrent Evaluation

$$\frac{\mathcal{C}, \mathcal{K}, e \hookrightarrow \mathcal{C}', \mathcal{K}', e' \qquad \pi \notin \mathrm{dom}(\mathcal{P})}{\mathcal{C}, \mathcal{K}, \mathcal{P} \cup \{\langle \pi, e \rangle\} \Rightarrow \mathcal{C}', \mathcal{K}', \mathcal{P} \cup \{\langle \pi, e' \rangle\}}$$

$$\frac{\pi \notin \mathrm{dom}(\mathcal{P}) \qquad \pi' \notin \mathrm{dom}(\mathcal{P}) \qquad \pi \neq \pi'}{\mathcal{C}, \mathcal{K}, \mathcal{P} \cup \{\langle \pi, E[\mathbf{spawn}\, v] \rangle\} \Rightarrow \mathcal{C}, \mathcal{K}, \mathcal{P} \cup \{\langle \pi, E[()] \rangle, \langle \pi', v() \rangle\}}$$

$$\frac{\mathcal{C}, \mathcal{P} \mapsto (\mathcal{P}', \mathcal{P}'')/\mathcal{A} \qquad \mathcal{C}' = \mathcal{C} \pm \{\mathcal{A} \mapsto \mathtt{true}\}}{\mathcal{C}, \mathcal{K}, \mathcal{P} \Rightarrow \mathcal{C}', \mathcal{K}, (\mathcal{P} \setminus \mathcal{P}') \cup \mathcal{P}''}$$

B.3.2: Rendezvous Projection

$$\frac{p_1 = \langle \pi_1, E_1[\mathbf{sync}\ ev_1] \rangle \qquad \cdots \qquad p_k = \langle \pi_k, E_k[\mathbf{sync}\ ev_k] \rangle \qquad C \vdash (ev_1, \ldots, ev_k) \rightsquigarrow_k (e_1, \ldots, e_k)/\mathcal{A}}{\mathcal{C}, \{p_1, \ldots, p_k\} \mapsto (\{p_1, \ldots, p_k\}, \{\langle \pi_1, E_1[e_1] \rangle, \ldots \langle \pi_k, E_k[e_k] \rangle\})/\mathcal{A}}$$

$$\frac{\mathcal{C}, \mathcal{P} \mapsto (\mathcal{P}', \mathcal{P}'')/\mathcal{A} \qquad \mathcal{P} \subseteq \mathcal{P}'''}{\mathcal{C}, \mathcal{P}''' \mapsto (\mathcal{P}', \mathcal{P}'')/\mathcal{A}}$$

B.3.3: Event Matching

$$\frac{}{C \vdash (\kappa\,!\,v, \kappa?) \rightsquigarrow_2 ((), v)/\emptyset}$$

$$\frac{\mathcal{C}(\gamma) = \texttt{true}}{\mathcal{C} \vdash \gamma \rightsquigarrow_1 ()/\emptyset}$$

Nack conditions:

$$\begin{aligned}
\texttt{nack}(ev_1 \oplus ev_2) &= \texttt{nack}(ev_1) \cup \texttt{nack}(ev_2), \\
\texttt{nack}(ev \Rightarrow v) &= \texttt{nack}(ev), \\
\texttt{nack}(ev \mid \gamma) &= \texttt{nack}(ev) \cup \{\gamma\}, \\
\texttt{nack}(ev) &= \emptyset, \text{ otherwise}
\end{aligned}$$

B.3.3: Event Matching (Cont.)

$$\frac{C \vdash (ev_1, \ldots, ev_k) \rightsquigarrow_k (e_1, \ldots, e_k)/\mathcal{A}}{C \vdash (ev_1 \oplus ev', \ldots, ev_k) \rightsquigarrow_k (e_1, \ldots, e_k)/\mathcal{A} \cup \mathtt{nack}(ev')}$$

$$\frac{C \vdash (ev_1, \ldots, ev_k) \rightsquigarrow_k (e_1, \ldots, e_k)/\mathcal{A}}{C \vdash (ev' \oplus ev_1, \ldots, ev_k) \rightsquigarrow_k (e_1, \ldots, e_k)/\mathcal{A} \cup \mathtt{nack}(ev')}$$

$$\frac{C \vdash (ev_1, \ldots, ev_k) \rightsquigarrow_k (e_1, \ldots, e_k)/\mathcal{A}}{C \vdash (ev_1 \Rightarrow f, \ldots, ev_k) \rightsquigarrow_k ((f\,e_1), \ldots, e_k)/\mathcal{A}}$$

$$\frac{C \vdash (ev_1, \ldots, ev_k) \rightsquigarrow_k (e_1, \ldots, e_k)/\mathcal{A}}{C \vdash (ev_1 \mid \gamma, \ldots, ev_k) \rightsquigarrow_k (e_1, \ldots, e_k)/\mathcal{A}}$$

$$\frac{C \vdash (ev_1, \ldots, ev_k) \rightsquigarrow_k (e_1, \ldots, e_k)/\mathcal{A} \qquad \{\, i_j \mid 1 \leq i_j \leq k \,\}}{C \vdash (ev_{i_1}, \ldots, ev_{i_k}) \rightsquigarrow_k (e_{i_1}, \ldots, e_{i_k})/\mathcal{A}}$$

B.3.4: Sequential Evaluation

$$
\begin{aligned}
\mathcal{C}, \mathcal{K}, E[b\, v] &\hookrightarrow \mathcal{C}, \mathcal{K}, E[\delta(b, v)], \\
\mathcal{C}, \mathcal{K}, E[(\mathbf{fn}\ x \Rightarrow e) v] &\hookrightarrow \mathcal{C}, \mathcal{K}, E[e[x \mapsto v]], \\
\mathcal{C}, \mathcal{K}, E[\mathbf{let}\ x = v\ \mathbf{in}\ e] &\hookrightarrow \mathcal{C}, \mathcal{K}, E[e[x \mapsto v]], \\
\mathcal{C}, \mathcal{K}, E[\texttt{alwaysEvt}\, v] &\hookrightarrow \mathcal{C} \pm \{\gamma \mapsto \texttt{true}\}, \mathcal{K}, E[\gamma \Rightarrow (\mathbf{fn}\ () \Rightarrow v)], \\
\mathcal{C}, \mathcal{K}, E[\texttt{channel}()] &\hookrightarrow \mathcal{C}, \mathcal{K} \cup \{\kappa\}, E[\kappa], \\
\mathcal{C}, \mathcal{K}, E[\mathbf{sync}(\mathbf{N}\, v)] &\hookrightarrow \mathcal{C} \pm \{\gamma \mapsto \texttt{false}\}, \mathcal{K}, E[\mathbf{sync}(v\, \gamma)],
\end{aligned}
$$

where $\gamma \notin \mathrm{dom}(\mathcal{C})$ and $\kappa \notin \mathcal{K}$.

$$\delta \in \mathrm{FConst} \times \mathrm{Val} \to \mathrm{Val}$$

B.3.4: Sequential Evaluation (Cont.)

$$\delta(\texttt{fst}, (v_1, v_2)) = v_1,$$

$$\vdots$$

$$\delta(\texttt{neverEvt}, ()) = \Lambda,$$
$$\delta(\texttt{recvEvt}, \kappa) = \kappa?,$$
$$\delta(\texttt{sendEvt}, (\kappa, v)) = k\,!\,v$$

$$\delta(\texttt{guard}, v) = \mathbf{N}(\mathbf{fn}\ a \Rightarrow v()),$$
$$\delta(\texttt{withNack}, v) = \mathbf{N}(\mathbf{fn}\ a \Rightarrow (v\,a \mid a)),$$

B.3.4: Sequential Evaluation (Cont.)

$$\delta(\texttt{wrap}, (\mathbf{N}\, v_1, v_2)) = \mathbf{N}(\mathbf{fn}\; a \Rightarrow \texttt{wrap}(v_1\, a, v_2)),$$
$$\delta(\texttt{wrap}, (ev, v)) = ev \Rightarrow v, \text{ otherwise}$$

$$\delta(\texttt{choose}, (\mathbf{N}\, v_1, \mathbf{N}\, v_2)) = \mathbf{N}(\mathbf{fn}\; a_1 \Rightarrow \mathbf{N}(\mathbf{fn}\; a_2 \Rightarrow \texttt{choose}(v_1\, a_1, v_2\, a_2))),$$
$$\delta(\texttt{choose}, (\mathbf{N}\, v, ev)) = \mathbf{N}(\mathbf{fn}\; a \Rightarrow \texttt{choose}(v\, a, ev)),$$
$$\delta(\texttt{choose}, (ev, \mathbf{N}\, v)) = \mathbf{N}(\mathbf{fn}\; a \Rightarrow \texttt{choose}(ev, v\, a)),$$
$$\delta(\texttt{choose}, (ev_1, ev_2)) = ev_1 \oplus ev_2, \text{ otherwise}$$

Execution Traces

A *trace* T is a (possibly infinite) sequence of well-formed configurations

$$T = \langle\langle \mathcal{C}_0, \mathcal{K}_0, \mathcal{P}_0; \mathcal{C}_1, \mathcal{K}_1, \mathcal{P}_1; \ldots \rangle\rangle$$

such that $\mathcal{C}_i, \mathcal{K}_i, \mathcal{P}_i \Rightarrow \mathcal{C}_{i+1}, \mathcal{K}_{i+1}, \mathcal{P}_{i+1}$ (for $i < n$, if T is finite with length n).

The *head* of T is $\mathcal{C}_0, \mathcal{K}_0, \mathcal{P}_0$.

Process States

Let $\mathcal{C}, \mathcal{K}, \mathcal{P}$ be a well-formed configuration, and let $\langle \pi, e \rangle \in \mathcal{P}$. The *state* of π in $\mathcal{P}$ is either *zombie*, *blocked*, *stuck* or *ready*, depending upon the form of e:

- if $e = [v]$, then π is a *zombie*;
- if $e = E[\mathbf{sync}\, ev]$ and there does not exist a set of processes

  $$\{ \langle \pi_i, E_i[\mathbf{sync}\, ev_i] \rangle \mid 1 \leq i \leq k \} \subseteq \mathcal{P} \setminus \{\langle \pi, e \rangle\}$$

  such that $\mathcal{C} \vdash (ev, ev_1, \ldots, ev_k) \rightsquigarrow_{k+1} (e', e_1, \ldots, e_k) \setminus \mathcal{A}$, then π is *blocked*;
- if there is no sequential configuration $\mathcal{C}, \mathcal{K}, e'$ such that $\mathcal{C}, \mathcal{K}, e \hookrightarrow \mathcal{C}, \mathcal{K}, e'$, then π is *stuck*;
- otherwise, π is *ready*.

Ready Processes

We define the set of ready processes in $\mathcal{C}, \mathcal{K}, \mathcal{P}$ by

$$\mathrm{Rdy}(\mathcal{C}, \mathcal{K}, \mathcal{P}) = \{\, \pi \mid \pi \text{ is ready in } \mathcal{C}, \mathcal{K}, \mathcal{P} \,\}.$$

A configuration $\mathcal{C}, \mathcal{K}, \mathcal{P}$ is *terminal* iff $\mathrm{Rdy}(P) = \emptyset$.

A terminal configuration with blocked processes is said to be *deadlocked*.

Computations

A trace is a *computation* iff it is maximal, i.e., it is infinite, or it is finite and ends in a terminal configuration.

If e is a program, then we define the computations of e to be

$$\mathrm{Comp}(e) = \{\, T \mid T \text{ is a computation with head } \emptyset, \emptyset, \{\langle \pi_0, e \rangle\} \,\}.$$

The *set of processes of* a trace T is defined as

$$\mathrm{Procs}(T) = \{\, \pi \mid \exists \mathcal{C}_i, \mathcal{K}_i, \mathcal{P}_i \in T \text{ with } \pi \in \mathrm{dom}(P_i) \,\}.$$

Fairness

The *synchronization objects* of an event value ev are defined as

$$\begin{aligned}
\mathrm{SyncObj}(\kappa!v) &= \{\kappa\},\\
\mathrm{SyncObj}(\kappa?) &= \{\kappa\},\\
\mathrm{SyncObj}(\gamma) &= \{\gamma\},\\
\mathrm{SyncObj}(\Lambda) &= \emptyset,\\
\mathrm{SyncObj}(ev_1 \oplus ev_2) &= \mathrm{SyncObj}(ev_1) \cup \mathrm{SyncObj}(ev_2),\\
\mathrm{SyncObj}(ev \Rightarrow v) &= \mathrm{SyncObj}(ev),\\
\mathrm{SyncObj}(ev \mid v) &= \mathrm{SyncObj}(ev).
\end{aligned}$$

We say that a synchronization object γ is *used* in a synchronization iff it is the synchronization object of the chosen base event for some process involved in the synchronization.

Fairness (Cont.)

A synchronization object ψ is *enabled* in a configuration $\mathcal{C}, \mathcal{K}, \mathcal{P}$ iff there are k processes $\langle \pi_i, E_i[\mathbf{sync}\, ev_i] \rangle \in \mathcal{P}$ for $1 \leq i \leq k$, such that $\psi \in \mathrm{SyncObj}(ev_1)$ and

$$\mathcal{C} \vdash (ev_1, \ldots, ev_k) \rightsquigarrow_k (e_1, \ldots, e_k) \setminus \mathcal{A}.$$

A computation T is *acceptable* iff it ends in a terminal configuration, or if T satisfies the following fairness constraints:

(1) Any process that is ready infinitely often is selected infinitely often.

(2) Any synchronization object that is enabled infinitely often is used infinitely often.

An implementation of CML should prohibit the possibility of unacceptable computations. In practice this requires that an implementation satisfy some stronger property on finite traces.