# Robot Control Language Semantics

April 18, 2004

This is the PS5 language specification.

## 1 Syntax

| | | |
|---|---|---|
| $n \in \mathbf{Z}$ | $=$ | $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$ |
| $loc \in \mathbf{Loc}$ | | Locations in memory—appear only in programs being evaluated |
| $aid \in \mathbf{Aid}$ | | Action identifiers—appear only in programs being evaluated |
| $v \in \mathbf{Value}$ | $::=$ | $n \mid (v_1, v_2) \mid \texttt{fn}\ id \Rightarrow e \mid loc$ |
| $e \in \mathbf{Expr}$ | $::=$ | $v \mid id \mid unop\ e_0 \mid e_0\ binop\ e_1 \mid e_0\ ;\ e_1 \mid action \mid \texttt{lref}\ e$ |
| | $\mid$ | $\texttt{gref}\ e \mid\ !e \mid e_0 := e1 \mid (\ e_0,\ e_1) \mid \texttt{let}\ id = e_0\ \texttt{in}\ e_1 \mid \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2$ |
| | $\mid$ | $e_1\ e_2 \mid \texttt{split}\ e_0\ \texttt{of}\ e_1\ \texttt{else}\ e_2 \mid aid$ |
| $binop$ | $::=$ | $+ \mid - \mid * \mid / \mid \texttt{mod} \mid < \mid =$ |
| $unop$ | $::=$ | $\sim \mid \texttt{rand} \mid \texttt{acquire} \mid \texttt{release} \mid \texttt{typeof}$ |
| $action$ | $::=$ | $\texttt{spawn}\ e \mid \texttt{do}\ e$ |

The operation `do` interacts with the external world; essentially it is the single I/O operation. The significance of particular inputs and outputs will be defined in PS6.

## 2 Operational semantics

### 2.1 Configuration

A memory $\sigma$ is a partial function from locations to values and process ids.

$$\sigma \in \mathbf{Memory} \quad = \quad \mathbf{Loc} \rightharpoonup \mathbf{Value} \times (\mathbf{Pid} \cup \emptyset)$$

The symbol $\rightharpoonup$ indicates a partial function that is defined only on some subset of **Loc** denoted by $\mathrm{dom}(\sigma)$. We write $\sigma[loc \mapsto (v, p)]$ to mean the function that is exactly like $\sigma$ except that it maps $loc$ to $(v, p)$, possibly extending the domain to include $loc$. The process idea stored at each memory location identifies which RCL program is currently holding a lock on that location. Locks are useful for controlling access to locations in memories shared by multiple RCL processes.

A configuration for a single process of the machine (that is, a single robot) is a pair $\langle e, \sigma \rangle$, and the configuration for the entire machine running $n$ concurrent processes is as follows:

$$\langle \sigma_g, \langle e_1, \sigma_1 \rangle, \ldots, \langle e_n, \sigma_n \rangle \rangle$$

where $\sigma_g$ is the shared memory between the processes, and $\sigma_m$ is the local memory for process $m$.

Every local memory has a special location *pid* where it stores the unique *process identifier* for the robot running the process. The process identifier is a positive integer. The process identifier is not the same thing as the index $m$, because the index of a process changes in the configuration as it evaluates.

### 2.2 Simple reductions

The following simple expression reductions do not involve memory and are similar to the evaluation reductions given for SML:

$$
\begin{array}{rcll}
\mathit{unop}\ v & \longrightarrow & v' & \text{where } v' = \underline{\mathit{unop}\ v} \\
v_0\ \mathit{binop}\ v_1 & \longrightarrow & v' & \text{where } v' = \underline{v_0\ \mathit{binop}\ v_1} \\
v;\ e & \longrightarrow & e & \\
\texttt{let}\ \mathit{id} = v\ \texttt{in}\ e & \longrightarrow & e\{v/\mathit{id}\} & \\
\texttt{if}\ v\ \texttt{then}\ e_1\ \texttt{else}\ e_2 & \longrightarrow & e_1 & \text{where } v \in Z^+ \\
\texttt{if}\ v\ \texttt{then}\ e_1\ \texttt{else}\ e_2 & \longrightarrow & e_2 & v \notin Z^+ \\
\texttt{fn}\ \mathit{id}\ \Rightarrow\ e(v) & \longrightarrow & e\{v/\mathit{id}\} & \\
\texttt{split}\ (v_1, v_2)\ \texttt{of}\ e_1\ \texttt{else}\ e_2 & \longrightarrow & (e_1\ v_1)\ v_2 & \\
\texttt{split}\ v\ \texttt{of}\ e_1\ \texttt{else}\ e_2 & \longrightarrow & e_2 & \text{where } v \text{ is not a pair} \\
\texttt{typeof}\ (\texttt{fn}\ \mathit{id}\ \Rightarrow\ e) & \longrightarrow & 0 & \\
\texttt{typeof}\ v & \longrightarrow & 1 & \text{where } v \in Z \\
\texttt{typeof}\ (v_1, v_2) & \longrightarrow & 2 & \\
\texttt{typeof}\ \mathit{loc} & \longrightarrow & 3 &
\end{array}
$$

If a simple reduction $e \longrightarrow e'$ is legal, then a process $\langle e, \sigma \rangle$ can step to $\langle e', \sigma \rangle$; the local memory is unchanged by the step.

In addition, a process $\langle e, \sigma \rangle$ with process id, $p$ , can perform the following reductions that involve memory:

$$
\begin{array}{rcll}
\langle \sigma_g, \langle \texttt{lref}\ v, \sigma \rangle \ldots \rangle & \longrightarrow & \langle \sigma_g, \langle \mathit{loc}, \sigma[\mathit{loc} \mapsto (v, \emptyset)] \ldots \rangle \rangle & \mathit{loc} \notin \mathrm{dom}(\sigma) \\
\langle \sigma_g, \langle !\mathit{loc}, \sigma \rangle \ldots \rangle & \longrightarrow & \langle \sigma_g, \langle v, \sigma \rangle \ldots \rangle & \sigma(\mathit{loc}) = (v, \emptyset) \text{ or } \sigma_g(\mathit{loc}) = (v, p') \\
\langle \sigma_g, \langle \mathit{loc}\ :=\ v, \sigma \rangle \ldots \rangle & \longrightarrow & \langle \sigma_g, \langle v, \sigma[\mathit{loc} \mapsto (v, \emptyset)] \rangle \ldots \rangle & \mathit{loc} \in \mathrm{dom}(\sigma) \\
\langle \sigma_g, \langle \mathit{loc}\ :=\ v, \sigma \rangle \ldots \rangle & \longrightarrow & \langle \sigma_g[\mathit{loc} \mapsto (v, p')] \rangle, \langle v, \sigma \rangle \ldots \rangle & \sigma_g(\mathit{loc}) \mapsto (v', p') \text{ and } p' \in \{\emptyset, p\} \\
\langle \sigma_g, \langle \texttt{acquire}\ \mathit{loc}, \sigma \rangle \ldots \rangle & \longrightarrow & \langle \sigma_g, \langle \mathit{loc}, \sigma \rangle \ldots \rangle & \mathit{loc} \in \mathrm{dom}(\sigma) \\
\langle \sigma_g, \langle \texttt{acquire}\ \mathit{loc}, \sigma \rangle \ldots \rangle & \longrightarrow & \langle \sigma_g[\mathit{loc} \mapsto (v, p)], \langle \mathit{loc}, \sigma \rangle \ldots \rangle & \sigma_g(\mathit{loc}) \mapsto (v, \emptyset) \\
\langle \sigma_g, \langle \texttt{acquire}\ \mathit{loc}, \sigma \rangle \ldots \rangle & \longrightarrow & \langle \sigma_g, \langle \texttt{acquire}\ \mathit{loc}, \sigma \rangle \ldots \rangle & \sigma_g(\mathit{loc}) \mapsto (v, p') \text{ and } p' \neq p \\
\langle \sigma_g, \langle \texttt{release}\ \mathit{loc}, \sigma \rangle \ldots \rangle & \longrightarrow & \langle \sigma_g, \langle \mathit{loc}, \sigma \rangle \ldots \rangle & \mathit{loc} \in \mathrm{dom}(\sigma) \\
\langle \sigma_g, \langle \texttt{release}\ \mathit{loc}, \sigma \rangle \ldots \rangle & \longrightarrow & \langle \sigma_g[\mathit{loc} \mapsto (v, \emptyset))], \langle \mathit{loc}, \sigma \rangle \ldots \rangle & \sigma_g(\mathit{loc}) \mapsto (v, p)
\end{array}
$$

### 2.3 Evaluation contexts

The following grammar defines where a reduction can occur during evaluation. It generates language terms in which there is a single hole $[\cdot]$ at the place where a reduction can be performed.

$$
\begin{array}{rcl}
\mathcal{C} & ::= & [\cdot] \mid \mathit{unop}\ \mathcal{C} \mid \mathcal{C}\ \mathit{binop}\ e \mid v\ \mathit{binop}\ \mathcal{C} \mid \mathcal{C};\ e \mid \texttt{do}\ \mathcal{C} \mid \texttt{spawn}\ \mathcal{C} \\
& \mid & \texttt{lref}\ \mathcal{C} \mid \texttt{gref}\ \mathcal{C} \mid !\,\mathcal{C} \mid \mathcal{C}\ :=\ e \mid \mathit{loc}\ :=\ \mathcal{C} \\
& \mid & \texttt{let}\ \mathit{id} = \mathcal{C}\ \texttt{in}\ e \mid (\mathcal{C}, e) \mid (v, \mathcal{C}) \mid \texttt{if}\ \mathcal{C}\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \\
& \mid & \mathcal{C}\ e \mid v\ \mathcal{C} \mid \texttt{split}\ \mathcal{C}\ \texttt{of}\ e_1\ \texttt{else}\ e_2
\end{array}
$$

Given a reduction $e \longrightarrow e'$, a single process can perform the evaluation step $\mathcal{C}[e] \longrightarrow \mathcal{C}[e']$, where $\mathcal{C}$ is a single evaluation context as described by the above grammar and $\mathcal{C}[e]$ represents that evaluation context with the term $e$ substituted for the single hole appearing in $\mathcal{C}$.

For example, the above grammar can derive the term with a hole $\texttt{let x} = [\cdot]\ \texttt{in}\ (\texttt{fn}\ (\texttt{y}) \Rightarrow \texttt{y})$, and $2 + 2 \longrightarrow 4$ is a legal reduction; therefore, the following is a legal single-process evaluation step:

$$\texttt{let x} = 2 + 2 \texttt{ in fn (y)} \Rightarrow \texttt{y} \quad \longrightarrow \quad \texttt{let x} = 4 \texttt{ in (fn (y)} \Rightarrow \texttt{y)}$$

### 2.4 Concurrency

As discussed earlier, the state of all the robot programs is represented by a global configuration

$$\langle \sigma_g, \langle e_1, \sigma_1 \rangle, \ldots, \langle e_n, \sigma_n \rangle \rangle$$

Steps of the machine therefore have the form

$$\langle \sigma_g, \langle e_1, \sigma_1 \rangle, \ldots, \langle e_n, \sigma_n \rangle \rangle \longrightarrow \langle \sigma'_g, \langle e'_1, \sigma'_1 \rangle, \ldots, \langle e'_{n'}, \sigma'_{n'} \rangle \rangle$$

To ensure round-robin scheduling, only the leftmost process takes a step, and the resulting process is always placed at the *end* of the list of processes. Thus, all of the single-process reduction rules can be lifted to apply to the entire machine using the following rule:

$$\frac{\langle e, \sigma_1 \rangle \longrightarrow \langle e', \sigma'_1 \rangle}{\langle \sigma_g, \langle \mathcal{C}[e], \sigma_1 \rangle, \langle e_2, \sigma_2 \rangle, \ldots, \langle e_n, \sigma_n \rangle \rangle \longrightarrow \langle \sigma_g, \langle e_2, \sigma_2 \rangle, \ldots, \langle e_n, \sigma_n \rangle, \langle \mathcal{C}[e'], \sigma'_1 \rangle \rangle}$$

A few more rules are needed for evaluation of expressions that access the that create or destroy a process:

$\langle \sigma_g, \langle \mathcal{C}[\texttt{spawn } v_s], \sigma_1 \rangle, \langle e_2, \sigma_2 \rangle, \ldots, \langle e_n, \sigma_n \rangle \rangle \longrightarrow \langle \sigma_g, \langle e_2, \sigma_2 \rangle, \ldots, \langle e_n, \sigma_n \rangle, \langle \mathcal{C}[e'_1], \sigma_1 \rangle, \langle v_s\ e'_2, \sigma_1[pid \mapsto p] \rangle \rangle$
where $p$ is a fresh process identifier and where $(e'_1, e'_2)$ is the response of the external world to the request $\texttt{Spawn}(\sigma_1(pid), p)$.

$\langle \sigma_g, \langle v_1, \sigma_1 \rangle, \langle e_2, \sigma_2 \rangle, \ldots, \langle e_n, \sigma_n \rangle \rangle \longrightarrow \langle \sigma_g, \langle e_2, \sigma_2 \rangle, \ldots, \langle e_n, \sigma_n \rangle \rangle$
Effect: send $\texttt{Halted}(\sigma_1(pid))$ to external world

## 3 Interaction with the external world

The last rule says that a process that has evaluated to a value simply disappears (it has halted). This rule and the rules for `spawn` and `do` require some communication with external world to inform it that a robot has been created, destroyed, or has taken some action. It is then up to the external world to give the robot(s) involved in the action an expression that is used in subsequent evaluation.

For example, in the `spawn` expression one natural choice for the expressions $e'_1$ and $e'_2$ is the process identifiers of the other robot. However, the external world may provide different information.

It may be necessary for the external world to delay execution of a process that has performed an action. It can achieve this by returning a value that is an action identifier *aid*. When asked to evaluate an action identifier, the interpreter always asks the external world to provide a value for it. This value then replaces the action identifier and evaluation proceeds. The external world can delay the interpreter as long as necessary by simply providing the same action identifier, causing the process to be placed on the back of the process queue unchanged.

| | | | |
|---|---|---|---|
| $\langle \texttt{do } v, \sigma \rangle$ | $\longrightarrow$ | $\langle e, \sigma \rangle$ | where $e$ is the result of $\texttt{Do}(\sigma(pid), v)$ |
| $\langle aid, \sigma \rangle$ | $\longrightarrow$ | $\langle aid, \sigma \rangle$ | if external world says no step may be taken |
| $\langle aid, \sigma \rangle$ | $\longrightarrow$ | $\langle e, \sigma \rangle$ | if external world responds to request $\texttt{ActionId}(\sigma(pid), aid)$ with result $e$. |

Because an action identifier *aid* is a legal expression, the second rule actually subsumes the first one.

It is up to the external world to keep track of the various action identifiers and what processes and actions they correspond to. The interpreter has no control over this aspect of evaluation.