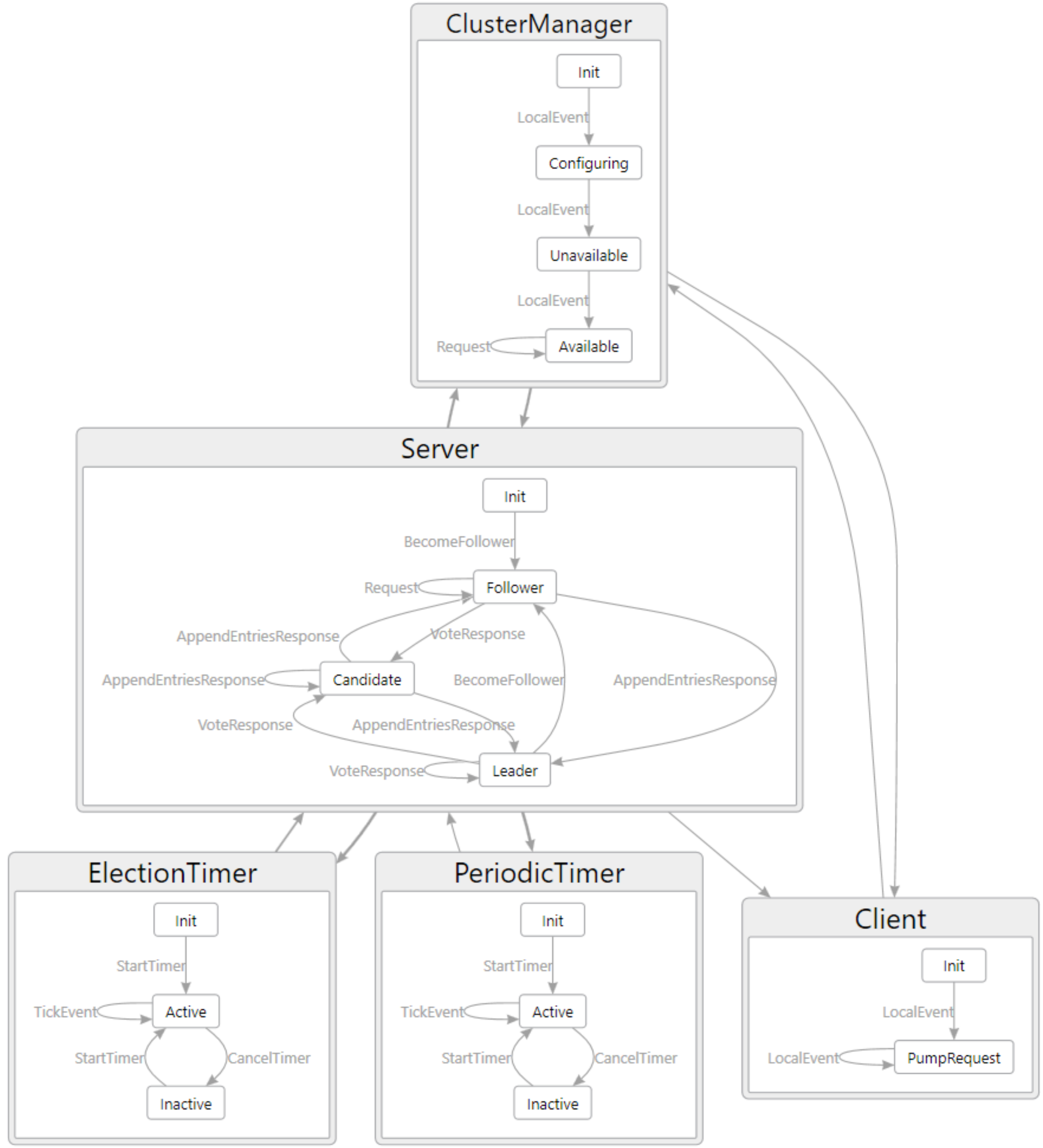
CS 428/528 Lecture 15: The P Framework for Communicating State Machines

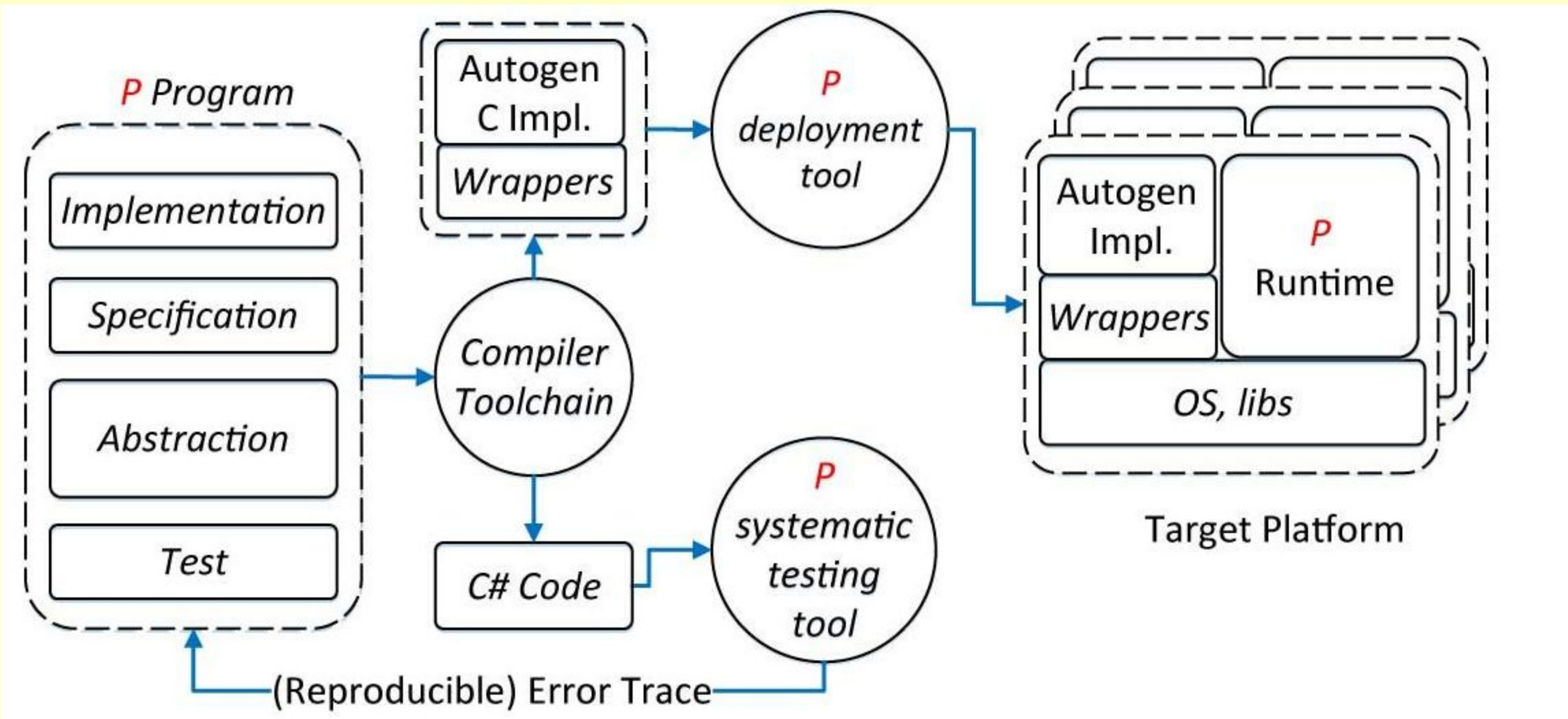
March 5, 2024

Based on the PLDI13 paper by Desai et al

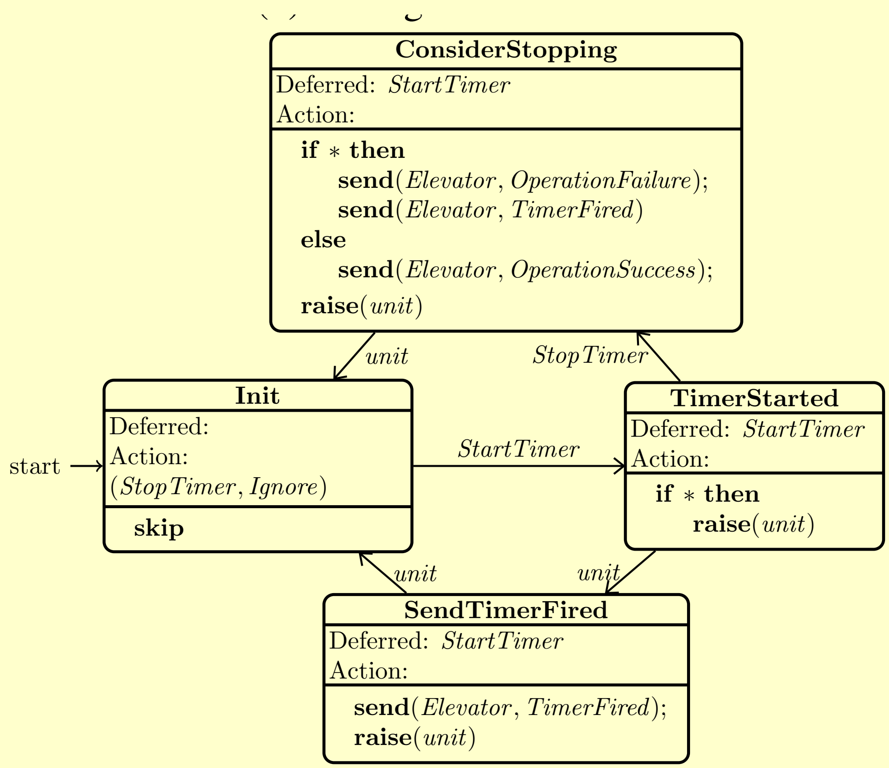
Communicating State Machines



P Architecture

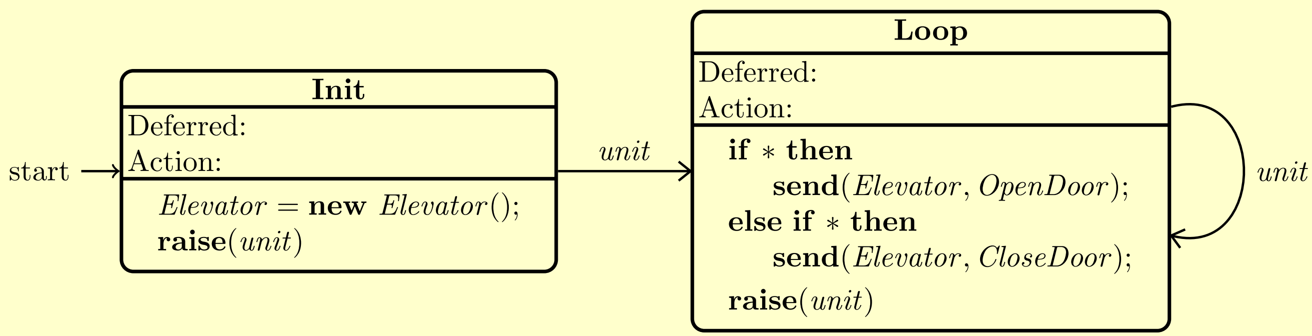


P Examples



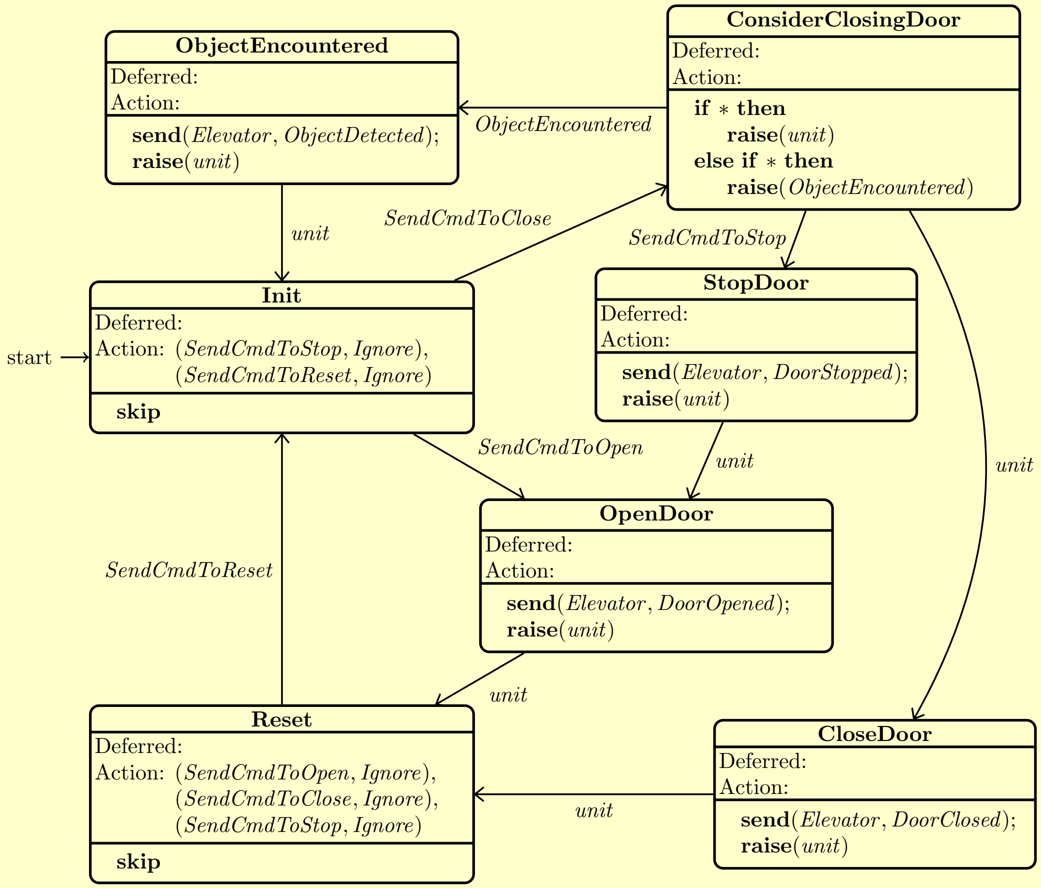
(c) `Timer` ghost machine

P Examples



(a) `User` ghost machine

P Examples



(b) `Door` ghost machine

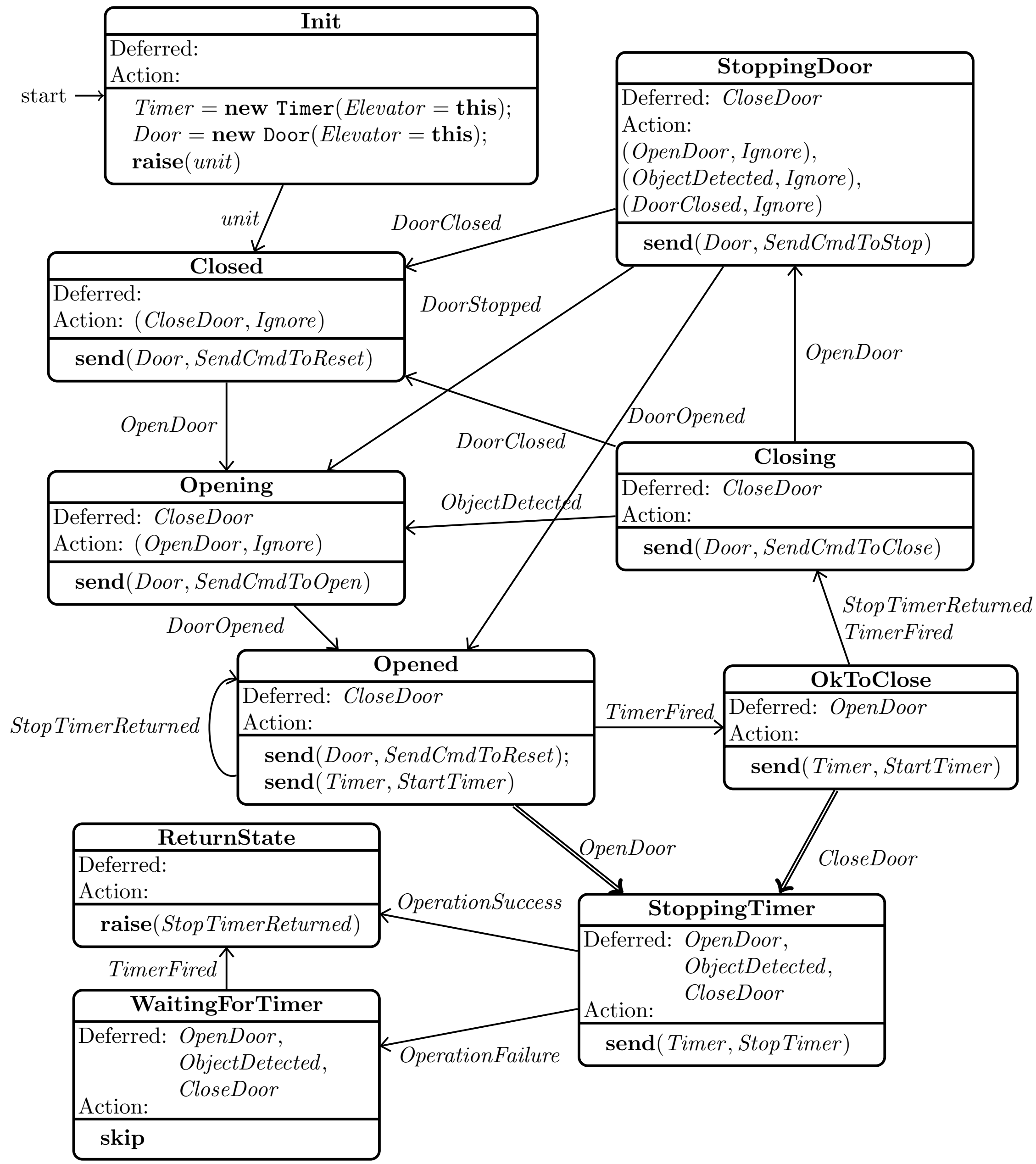


Figure 1: Elevator example

P Language Syntax

program	::=	evdecl machine$^+$ m(init*)
machine	::=	optghost **machine** m vrdecl* actdecl* stdecl* spdecl* cldecl* acdecl*
optghost	::=	ϵ | **ghost**
evdecl	::=	**event** edecl$^+$
vrdecl	::=	optghost **var** vdecl$^+$
actdecl	::=	**action** $(a$, stmt$)^+$
stdecl	::=	**state** $(n, \{e_1, e_2, \ldots, e_k\}$, stmt, stmt$)^+$
spdecl	::=	**step** $(n, e, n)^+$
cldecl	::=	**call** $(n, e, n)^+$
acdecl	::=	**act** $(n, e, a)^+$
edecl	::=	e(type)
vdecl	::=	x : type
type	::=	**void** | **bool** | **int** | **event** | **id**

stmt	::=	**skip**
	|	x := expr
	|	x := **new** m(init*)
	|	**delete**
	|	**send**(expr, e, expr)
	|	**raise** (e, expr)
	|	**leave**
	|	**return**
	|	**assert**(expr)
	|	stmt; stmt
	|	**if** expr **then** stmt **else** stmt
	|	**while** expr stmt
init	::=	x = expr
expr	::=	**this** | **msg** | **arg** | b | c | $\perp$ | x | $*$ | uop expr | expr bop expr

$c \in$ **int**	$b \in$ **bool**
$\neg, - \in$ uop	$+, -, \wedge, \vee \in$ bop
$r \in$ expr	$a, e, m, x \in$ name

P Language Semantics

configuration. A machine configuration corresponding to identifier id is of the form (γ, σ, s, q) with components defined as follows:

- γ is a sequence of pairs (n, α), where n is a state name, and α is map from events to $A \cup \{\top, \bot\}$, where A is the set of all actions declared in machine $Name(id)$. This sequence functions as a call stack, to implement call and return, and the α values are used to inherit deferred events and actions from caller to callee. For an event e, $\alpha(e)$ can be an action a, or the value $\top$ indicating that the event is deferred, or the value $\bot$ which indicates that the event does not have an associated action and it is not deferred.
- σ is a map from variables declared in machine $Name(id)$ to their values; this map contains an entry for the local variables **this**, **msg** and **arg**.
- s is the statement remaining to be executed in machine id.
- q is a sequence of pairs of a event-argument pairs representing the input buffer of machine id.

P Language Semantics

state n is executed whenever control leaves n. Given a machine name m and a state n in m, let $Deferred(m, n)$ denote the associated set of deferred events and let $Action(m, n, e)$ be an that action a is associated with event e in state n, if such a binding exists or $\perp$ otherwise. Let $Entry(m, n)$ denote the associated entry statement, and let $Exit(m, n)$ denote the associated exit statement. The initial state of the machine m is the first state in the state list and is denoted by $Init(m)$.

P Language Semantics

$$\frac{M[id] = (\gamma, \sigma, S[x := r], q) \qquad \sigma(r) \downarrow v}{M \longrightarrow M[id := (\gamma, \sigma[x := v], S[\mathbf{skip}], q)]} \; (\textsc{Assign})$$

$$\frac{\begin{array}{c} M[id] = (\gamma, \sigma, S[x := \mathbf{new}\; m'(x_1 = r_1, x_2 = r_2, \ldots, x_n = r_n)], q) \\ id' = \mathit{fresh}(m') \qquad n' = \mathit{Init}(m') \\ \alpha_o = \lambda e.\perp \qquad \sigma(r_1) \downarrow v_1 \qquad \sigma(r_2) \downarrow v_2 \qquad \cdots \qquad \sigma(r_n) \downarrow v_n \\ \sigma' = \lambda x.\perp [\mathbf{this} := id'][x_1 := v_1][x_2 := v_2] \cdots [x_n := v_n] \end{array}}{\begin{array}{c} M \longrightarrow M[id := (\gamma, \sigma[x := id'], S[\mathbf{skip}], q)] \\ \qquad\qquad [id' := ((n', \alpha_o), \sigma', \mathit{Entry}(m', n'), \epsilon)] \end{array}} \; (\textsc{New})$$

$$\frac{M[id] = (\gamma, \sigma, S[\mathbf{delete}], q)}{M \longrightarrow M[id := \perp]} \; (\textsc{Delete})$$

$$\frac{M[id] = (\gamma, \sigma, S[\mathbf{assert}(r)], q) \qquad \sigma(r) \downarrow \mathbf{true}}{M \longrightarrow M[id := (\gamma, \sigma, S[\mathbf{skip}], q)]} \; (\textsc{Assert-Pass})$$

$$\frac{M[id] = (\gamma, \sigma, S[\mathbf{skip}; s], q)}{M \longrightarrow M[id := (\gamma, \sigma, S[s], q)]} \; (\textsc{Seq})$$

P Language Semantics

$$\frac{M[id] = (\gamma, \sigma, S[\textbf{if } r \textbf{ then } s_1 \textbf{ else } s_2], q) \qquad \sigma(r) \downarrow \textbf{true}}{M \longrightarrow M[id := (\gamma, \sigma, S[s_1], q)]} \ (\textsc{If-Then})$$

$$\frac{M[id] = (\gamma, \sigma, S[\textbf{if } r \textbf{ then } s_1 \textbf{ else } s_2], q) \qquad \sigma(r) \downarrow \textbf{false}}{M \longrightarrow M[id := (\gamma, \sigma, S[s_2], q)]} \ (\textsc{If-Else})$$

$$\frac{M[id] = (\gamma, \sigma, S[\textbf{while } r \ s], q) \qquad \sigma(r) \downarrow \textbf{true}}{M \longrightarrow M[id := (\gamma, \sigma, S[s; \textbf{while } r \ s], q)]} \ (\textsc{While-Iterate})$$

$$\frac{M[id] = (\gamma, \sigma, S[\textbf{while } r \ s], q) \qquad \sigma(r) \downarrow \textbf{false}}{M \longrightarrow M[id := (\gamma, \sigma, S[\textbf{skip}], q)]} \ (\textsc{While-Done})$$

$$\frac{\begin{array}{c} M[id] = (\gamma, \sigma, S[\textbf{send}(r_1, e, r_2)], q) \\ \sigma(r_1) \downarrow id' \qquad \sigma(r_2) \downarrow v \qquad M[id'] = (\gamma', \sigma', C', q') \end{array}}{M \longrightarrow M[id := (\gamma, \sigma, S[\textbf{skip}], q)][id' := (\gamma', \sigma', C', q' \odot (e, v))]} \ (\textsc{Send})$$

P Language Semantics

$$\frac{\begin{array}{c} M[id] = ((n,\alpha)\cdot\gamma, \sigma, S[\textbf{raise}\ (e,r)], q) \\ \sigma(r) \downarrow v \qquad \sigma' = \sigma[\textbf{msg} := e][\textbf{arg} := v] \qquad m = Name(id) \\ \begin{array}{rl} s = & \textbf{if}\ Pop(m,n,\alpha,e) \vee Step(m,n,e) \neq \perp \\ & \textbf{then}\ Exit(m,n) \\ & \textbf{else skip} \end{array} \end{array}}{M \longrightarrow M[id := ((n,\alpha)\cdot\gamma, \sigma', s; \overline{\textbf{raise}}\ (e,v), q)]}\ (\textsc{Raise})$$

$$\frac{M[id] = (\gamma, \sigma, S[\textbf{leave}], q)}{M \longrightarrow M[id := (\gamma, \sigma, \textbf{skip}, q)]}\ (\textsc{Leave})$$

$$\frac{M[id] = (\gamma, \sigma, S[\textbf{return}], q)}{M \longrightarrow M[id := (\gamma, \sigma, Exit(m,n); \overline{\textbf{return}}, q)]}\ (\textsc{Return})$$

$$\frac{\begin{array}{c} M[id] = ((n,\alpha)\cdot\gamma, \sigma, \textbf{skip}, q_1\cdot(e,v)\cdot q_2) \qquad m = Name(id) \\ t = \{e \mid Trans(m,n,e) \neq \perp \vee Action(m,n,e) \neq \perp\} \\ d = \{e \mid \alpha(e) = \top\} \qquad d' = (d \cup Deferred(m,n)) - t \\ |q_1| \subseteq d' \qquad e \notin d' \qquad \sigma' = \sigma[\textbf{msg} := e][\textbf{arg} := v] \\ \begin{array}{rl} s = & \textbf{if}\ Pop(m,n,\alpha,e) \vee Step(m,n,e) \neq \perp \\ & \textbf{then}\ Exit(m,n) \\ & \textbf{else skip} \end{array} \end{array}}{M \longrightarrow M[id := ((n,\alpha)\cdot\gamma, \sigma', s; \overline{\textbf{raise}}\ (e,v), q_1\cdot q_2)]}\ (\textsc{Dequeue})$$

P Language Semantics

$$\frac{\begin{array}{c} M[id] = ((n,\alpha)\cdot\gamma, \sigma, \overline{\textbf{raise}}\ (e,v), q) \\ m = \mathit{Name}(id) \qquad \mathit{Step}(m,n,e) = n' \end{array}}{M \longrightarrow M[id := ((n',\alpha)\cdot\gamma, \sigma, \mathit{Entry}(m,n'), q)]}\ (\textsc{Step})$$

$$\frac{\begin{array}{c} M[id] = ((n,\alpha)\cdot\gamma, \sigma, \overline{\textbf{raise}}\ (e,v), q) \\ m = \mathit{Name}(id) \qquad \mathit{Trans}(m,n,e) = \perp \\ (\alpha(e) = a \wedge \mathit{Action}(m,n,e) = \perp) \vee \mathit{Action}(m,n,e) = a \\ a \notin \{\perp, \top\} \end{array}}{M \longrightarrow M[id := ((n,\alpha)\cdot\gamma, \sigma, \mathit{Stmt}(m,a), q)]}\ (\textsc{Action})$$

$$\frac{\begin{array}{ll} & M[id] = ((n,\alpha)\cdot\gamma, \sigma, \overline{\textbf{raise}}\ (e,v), q) \\ & m = \mathit{Name}(id) \qquad \mathit{Call}(m,n,e) = n' \\ \alpha' = \lambda e. & \textbf{if } (\mathit{Trans}(m,n,e) \neq \perp) \textbf{ then } \perp \\ & \textbf{else if } (\mathit{Action}(m,n,e) \neq \perp) \textbf{ then } \mathit{Action}(m,n,e) \\ & \textbf{else if } (e \in \mathit{Deferred}(m,n)) \textbf{ then } \top \\ & \textbf{else } \alpha(e) \end{array}}{M \longrightarrow M[id := ((n',\alpha')\cdot(n,\alpha)\cdot\gamma, \sigma, \mathit{Entry}(m,n'), q)]}\ (\textsc{Call})$$

$$\frac{\begin{array}{c} M[id] = ((n,\alpha)\cdot\gamma, \sigma, \overline{\textbf{raise}}\ (e,v), q) \\ m = \mathit{Name}(id) \qquad \mathit{Pop}(m,n,\alpha,e) \end{array}}{M \longrightarrow M[id := (\gamma, \sigma, \textbf{raise}\ (e,v), q)]}\ (\textsc{Pop1})$$

$$\begin{array}{rl} \mathit{Pop}(m,n,\alpha,e) = & \mathit{Step}(m,n,e) = \perp \wedge \\ & \mathit{Call}(m,n,e) = \perp \wedge \\ & \mathit{Action}(m,n,e) = \perp \wedge \\ & \alpha(e) \in \{\perp, \top\} \end{array}$$

$$\frac{M[id] = ((n,\alpha)\cdot\gamma, \sigma, \overline{\textbf{return}}, q) \qquad m = \mathit{Name}(id)}{M \longrightarrow M[id := (\gamma, \sigma, \textbf{skip}, q)]}\ (\textsc{Pop2})$$

P Language Semantics

$$\frac{M[id] = (\gamma, \sigma, S[\mathbf{assert}(r)], q) \qquad \sigma(r) \downarrow \mathbf{false}}{M \longrightarrow error} \ (\textsc{Assert-Fail})$$

$$\frac{M[id] = (\gamma, \sigma, S[\mathbf{send}(r_1, e, r_2)], q) \qquad \sigma(r_1) \downarrow \perp}{M \longrightarrow error} \ (\textsc{Send-Fail1})$$

$$\frac{\begin{array}{c} M[id] = (\gamma, \sigma, S[\mathbf{send}(r_1, e, r_2)], q) \\ \sigma(r_1) \downarrow id' \qquad M[id'] = \perp \end{array}}{M \longrightarrow error} \ (\textsc{Send-Fail2})$$

$$\frac{M[id] = (\epsilon, \sigma, s, q)}{M \longrightarrow error} \ (\textsc{Pop-Fail})$$

Figure 6: Operational semantics: error transitions