

## 1 Auxiliary Functions

`body(xs,m)` returns the body of message `m` in software actor `xs`

`highest_priority_message(Q)` returns the highest priority message from the given queue.

`highest_priority_packet(B)` returns the highest priority packet (sender-message-receiver tuple) from the given buffer.

`acquire_timer(p)` acquires a free timer variable from the give variable pool.

`release_timer(t,p)` releases timer variable `t` to the give variable pool.

`is_direct(x,y)` returns true if the `x` communicates with `y` directly (e.g. through wire), returns false otherwise.

`is_nat(x,y)` returns true if the `x` communicates with `y` through NAT, returns false otherwise.

`guard(xp,m)` returns the guard for mode `m` in the physical actor `xp`

`actions(xp,m)` returns the actions for mode `m` in the physical actor `xp`

`network_delay(x,m,y)` returns the network delay for sending message `m` from actor `x` to actor `y`

`no_software_action(gs)` returns true if there is no outgoing software action (transition) in state `gs`.

`no_network_action(gs)` returns true if there is no outgoing network action (transition) in state `gs`.

`execute(acts,gs)` returns global states by executing the given actions on the global state `gs`.

NOTE: It's not function. It is possible to have multiple global states by executing the actions.

## 2 Semantics

### 2.1 Global State

Global state `GS` is defined as a tuple `(DS,CS,NS,ES)`.

`DS` is the discrete state that is the states of all software actors.

`CS` is the continuous state that is the states of all physical actors.

`NS` is the network states that is tuple `(B,i)` where `B` is the buffer of current packets and `i` is a boolean which show where network is idle (ready to send) or not

`ES` is the events state that is the list of current events and the pool of timer variables.

The state of a physical actor is its current mode. A mode defines the invariant, ODE, guard and actions.

An event is a tuple (d,acts) where d is the delay of this event and acts are the actions that are executed when the event is fired.

$\xrightarrow{s}$  is software action

$\xrightarrow{n}$  is network action

$\xrightarrow{p}$  is physical action

## 2.2 Coarse-Grained Rules

Message Execution (FIFO)

(Labels must be retained for in execute )

$$\frac{DS(x) = (v, m|T, \epsilon) \wedge \neg v(suspended)}{GS \xrightarrow[s]{labels} execute(body(x, m), GS[DS \mapsto DS'])} \quad (1)$$

$$DS' = DS[x \mapsto (v, T, \epsilon)]$$

Message Execution (Priority-based)

$$\frac{DS(x) = (v, T, \epsilon) \wedge \neg v(suspended) \quad T \neq \epsilon \wedge m = highest\_priority\_message(T)}{GS \xrightarrow[s]{labels} execute(body(x, m), GS[DS \mapsto DS'])} \quad (2)$$

$$DS' = DS[x \mapsto (v, T', \epsilon)]$$

$$T' = T \setminus m$$

Continuous Behavior Expiration

$$\frac{CS(x) = m \wedge m \neq none \quad no\_software\_action(GS) \wedge no\_network\_action(GS)}{GS \xrightarrow[p]{guard(x, m)} execute(actions(x, m), GS[CS \mapsto CS'])} \quad (3)$$

$$CS' = CS[x \mapsto none]$$

CAN Bus

$$\begin{array}{c}
NS = (B, i) \wedge i = \text{true} \wedge \text{no\_software\_action}(GS) \\
(x, m, y) = \text{highest\_priority\_packet}(B) \\
d = \text{network\_delay}(x, m, y) \\
ES = (\text{events}, p) \\
\hline
GS \xrightarrow[n]{\quad} GS[NS \mapsto NS', ES \mapsto ES'] \\
ES = ES[\text{events} \mapsto \text{events} \oplus (d, \{\text{send\_direct}(y, m), i = \text{true}\})] \\
NS' = NS[B \mapsto B \setminus (x, m, y), i \mapsto \text{false}]
\end{array} \tag{4}$$

Events (How does the variable pool changes?)

$$\begin{array}{c}
TS = (\text{events}, p) \wedge (d, \text{acts}) \in \text{events} \\
\text{no\_software\_action}(GS) \wedge \text{no\_network\_action}(GS) \\
\text{timer} = \text{acquire\_timer}(p) \\
\hline
GS \xrightarrow[c]{\text{timer}=d, \text{timer}:=0} \text{execute}(\text{acts}, GS[ES \mapsto ES']) \\
ES' = [\text{events} \mapsto \text{events} \setminus (y, \text{acts}), p \mapsto \dots]
\end{array} \tag{5}$$

## 2.3 Fine-Grained Rules

Go To

$$\begin{array}{c}
DS(x) = (v, q, (p \text{ goto } m') | \sigma) \\
CS(p) = m \\
\hline
GS \xrightarrow{\tau} GS[DS \mapsto DS', CS \mapsto CS'] \\
DS' = DS[x \mapsto (c, q, \sigma)] \\
CS' = CS[p \mapsto m']
\end{array} \tag{6}$$

Delay Statement

$$\begin{array}{c}
DS(x) = (v, q, (\text{delay}(d) | \sigma)) \\
ES = (\text{events}, p) \\
\hline
GS \xrightarrow{\tau} GS[DS \mapsto DS', TS \mapsto ES'] \\
DS' = DS[x \mapsto (v[\text{suspended} \mapsto \text{true}], q, \sigma)] \\
ES' = ES[\text{events} \mapsto \text{events} \oplus (d, \{\text{resume}(x)\})]
\end{array} \tag{7}$$

Continuous Variable Assignment

$$\frac{\dots}{GS \xrightarrow{\text{cvar}=\text{eval}(\text{expr})} GS'} \tag{8}$$

Discrete Variable Assignment

$$\frac{DS(x) = (v, q, (dvar = expr|\sigma))}{GS \xRightarrow{\tau} GS[DS \mapsto DS']} \quad (9)$$

$$DS' = DS[x \mapsto (v[dvar \mapsto eval(expr)], q, \sigma)]$$

Conditional True

$$\frac{DS(x) = (v, q, (if \ expr \ \sigma \ else \ \sigma'|\sigma'')) \quad eval(expr) = True}{GS \xRightarrow{\tau} GS[DS \ longmapsto DS']} \quad (10)$$

$$DS' = DS[x \mapsto (v, q, \sigma \oplus \sigma'')]$$

Conditional False

$$\frac{DS(x) = (v, q, (if \ expr \ \sigma \ else \ \sigma'|\sigma'')) \quad eval(expr) = False}{GS \xRightarrow{\tau} GS[DS \mapsto DS']} \quad (11)$$

$$DS' = DS[x \mapsto (v, q, \sigma' \oplus \sigma'')]$$

Direct Message Send (Software Actor)

$$\frac{DS(x) = (v_x, q_x, ((y, m)|\sigma_x)) \quad DS(y) = (v_y, q_y, \sigma_y) \quad is\_direct(x, y)}{GS \xRightarrow{\tau} GS[DS \mapsto DS']} \quad (12)$$

$$DS' = DS[x \mapsto (v_x, q_x, \sigma_x), y \mapsto (v_y, q_y \oplus m, \sigma_y)]$$

NAT Message Send (Software Actor)

$$\frac{DS(x) = (v_x, q_x, ((y, m)|\sigma_x)) \quad NS = (B, ii) \quad is\_nat(x, y)}{GS \xRightarrow{\tau} GS[DS \mapsto DS', NS \mapsto NS']} \quad (13)$$

$$DS' = DS[x \mapsto (v_x, q_x, \sigma_x)]$$

$$NS' = NS[B \mapsto B \oplus (x, y, n)]$$

Direct Message Send (Physical Actor)

$$\frac{\dots}{G \xRightarrow{\tau} G[DS \mapsto DS']} \quad (14)$$

$$DS' = DS[x \mapsto (v_x, q_x, \sigma_x), y \mapsto (v_y, q_y \oplus m, \sigma_y)]$$

NAT Message Send (Physical Actor)

$$\frac{\dots}{GS \xRightarrow{\tau} GS[DS \mapsto DS', NS \mapsto NS']} \quad (15)$$

$$NS' = NS[B \mapsto B \oplus (x, m, y)]$$

### 3 Issues Regarding Formalization of Semantics

General statement execution: Currently most statements are define for software actors. But there some statements that are common between software actors and physical actors (e.g. send). Is there any way to statements' semantics generally?

Executing physical actors statements: In software actors we have a execution queue for executing software statements. Should we put a execution queue for physical actors too?

Continuous Message Parameters : How should we deal with continuous parameters in messages?

Detecting Concurrent Continuous behavior expiration

Defining the semantics of  $execute(acts, gs)$