# 1 Auxiliary Functions

body($x_s$,m) returns the body of message m in software actor $x_s$
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.

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

guard($x_p$,m) returns the guard for mode m in the physical actor $x_p$
actions($x_p$,m) returns the actions for mode m in the physical actor $x_p$

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.

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 and execution queue. A mode

defines the invariant, ODE, guard and actions.

An event is a tuple (d,event) where d is the delay of this event and "event" is the event that is executed .

$\underset{s}{\rightarrow}$ is software action
$\underset{n}{\rightarrow}$ is network action
$\underset{p}{\rightarrow}$ is physical action

$$\begin{gathered} effect(CAN\_EVENT(y,m),GS) = GS[DS \longmapsto DS', NS \longmapsto NS'] \\ DS' = DS[y \longmapsto y[q \longmapsto q \oplus m]] \\ NS' = NS[i \longmapsto true] \end{gathered} \tag{1}$$

$$\begin{gathered} effect(RESUME\_EVENT(x),GS) = GS[DS \longmapsto DS'] \\ DS' = DS[x \longmapsto x[v \longmapsto v[suspended \longmapsto false]]] \end{gathered} \tag{2}$$

## 2.2 SOS Rules

Message Take (FIFO)

$$\begin{gathered} \frac{DS(x) = (v, m|T, \epsilon) \wedge \neg v(suspended)}{GS \underset{s}{\rightarrow} GS[DS \longmapsto DS']} \\ DS' = DS[x \longmapsto (v, T, body(x,m))] \end{gathered} \tag{3}$$

Message Take (Priority-based)

$$\begin{gathered} \frac{\begin{gathered} DS(x) = (v, T, \epsilon) \wedge \neg v(suspended) \\ T \neq \epsilon \wedge m = highest\_priority\_message(T) \end{gathered}}{GS \underset{s}{\rightarrow} GS[DS \longmapsto DS']} \\ DS' = DS[x \longmapsto (v, T', body(x,m))] \\ T' = T \backslash m \end{gathered} \tag{4}$$

Continuous Behavior Expiration

$$\begin{gathered} \frac{\begin{gathered} CS(x) = (m, \epsilon) \wedge m \neq none \\ no\_software\_action(GS) \wedge no\_network\_action(GS) \end{gathered}}{GS \underset{p}{\xrightarrow{guard(x,m)}} GS[CS \longmapsto CS']} \\ CS' = CS[x \longmapsto (m, actions(m))] \end{gathered} \tag{5}$$

CAN

$$\frac{\begin{array}{c} NS = (B, i) \wedge i = true \wedge no\_software\_action(GS) \\ (x, m, y) = highest\_priority\_packet(B) \\ d = network\_delay(x, m, y) \\ ES = (events, p) \wedge timer = acquire\_timer(p) \end{array}}{\begin{array}{c} GS \underset{n}{\rightarrow} GS[NS \longmapsto NS', ES \longmapsto ES'] \\ ES = ES[events \longmapsto events \oplus (d, timer, CAN\_EVENT(y, m)), p \longmapsto p \backslash timer] \\ NS' = NS[B \longmapsto B \backslash (x, m, y), i \longmapsto false] \end{array}} \tag{6}$$

Events

$$\frac{\begin{array}{c} TS = (events, p) \wedge (d, timer, event) \in events \\ no\_software\_action(GS) \wedge no\_network\_action(GS) \end{array}}{\begin{array}{c} GS \xrightarrow[c]{timer=d, timer:=0} effect(event, GS[ES \longmapsto ES']) \\ ES' = [events \longmapsto events \backslash (d, timer, event), p \longmapsto p \oplus timer] \end{array}} \tag{7}$$

Go To

$$\frac{\begin{array}{c} DS(x) = (v, q, (p\ goto\ m') | \sigma_x) \\ CS(p) = (m, \sigma_p) \end{array}}{\begin{array}{c} GS \underset{s}{\rightarrow} GS[DS \longmapsto DS', CS \longmapsto CS'] \\ DS' = DS[x \longmapsto (c, q, \sigma_x)] \\ CS' = CS[p \longmapsto (m', \sigma_p)] \end{array}} \tag{8}$$

Delay Statement

$$\frac{\begin{array}{c} DS(x) = (v, q, (delay(d) | \sigma)) \\ ES = (events, p) \wedge timer = acquire\_timer(p) \end{array}}{\begin{array}{c} GS \underset{s}{\rightarrow} GS[DS \longmapsto DS', ES \longmapsto ES'] \\ DS' = DS[x \longmapsto (v[suspended \longmapsto true], q, \sigma)] \\ ES' = ES[events \longmapsto events \oplus (d, timer, RESUME\_EVENT(x)), p \longmapsto p \backslash timer] \end{array}} \tag{9}$$

Continuous Variable Assignment

$$\frac{CS(x) = (\theta, cvar = expr|\sigma)}{GS \xrightarrow[s]{cvar=eval(expr)} GS[CS \longmapsto CS']} \quad CS' = CS[x \longmapsto (\theta, \sigma)] \tag{10}$$

Discrete Variable Assignment

$$\frac{DS(x) = (v, q, (dvar = expr|\sigma))}{GS \underset{s}{\rightarrow} GS[DS \longmapsto DS']} \quad DS^{'} = DS[x \longmapsto (v[dvar \longmapsto eval(expr)], q, \sigma)] \tag{11}$$

Conditional True

$$\frac{\begin{array}{c} DS(x) = (v, q, (if\ expr\ \sigma\ else\ \sigma^{'}|\sigma^{''})) \\ eval(expr) = True \end{array}}{GS \underset{s}{\rightarrow} GS[DS\ longmapstoDS']} \quad DS^{'} = DS[x \longmapsto (v, q, \sigma \oplus \sigma^{''})] \tag{12}$$

Conditional False

$$\frac{\begin{array}{c} DS(x) = (v, q, (if\ expr\ \sigma\ else\ \sigma'|\sigma'')) \\ eval(expr) = False \end{array}}{GS \underset{s}{\rightarrow} GS[DS \longmapsto DS']} \quad DS^{'} = DS[x \longmapsto (v, q, \sigma' \oplus \sigma'')] \tag{13}$$

Direct Message Send (Software Actor)

$$\frac{\begin{array}{c} DS(x) = (v_x, q_x, ((y, m)|\sigma_x)) \\ DS(y) = (v_y, q_y, \sigma_y) \\ is\_direct(x, y) \end{array}}{GS \underset{s}{\rightarrow} GS[DS \longmapsto DS']} \quad DS^{'} = DS[x \longmapsto (v_x, q_x, \sigma_x), y \longmapsto (v_y, q_y \oplus m, \sigma_y)] \tag{14}$$

CAN Message Send (Software Actor)

$$\frac{\begin{array}{c} DS(x) = (v_x, q_x, ((y, m)|\sigma_x)) \\ NS = (B, i) \\ is\_can(x, y) \end{array}}{\begin{array}{c} GS \underset{s}{\rightarrow} GS[DS \longmapsto DS', NS \longmapsto NS'] \\ DS^{'} = DS[x \longmapsto (v_x, q_x, \sigma_x)] \\ NS' = NS[B \longmapsto B \oplus (x, y, n)] \end{array}} \tag{15}$$

Direct Message Send (Physical Actor)

$$\frac{\begin{array}{c} CS(x) = (\theta_x, (y, m)|\sigma_x) \\ DS(y) = (v_y, q_y, \sigma_y) \\ is\_direct(x, y) \end{array}}{\begin{array}{c} GS \underset{s}{\rightarrow} GS[DS \longmapsto DS', CS \longmapsto CS'] \\ DS' = DS[y \longmapsto (v_y, q_y \oplus m, \sigma_y)] \\ CS' = CS[x \longmapsto (\theta_x, \sigma_x)] \end{array}} \tag{16}$$

CAN Message Send (Physical Actor)

$$\frac{CS(x) = (\theta, (y, m)|\sigma) \wedge is\_can(x, y)}{\begin{array}{c} GS \underset{s}{\rightarrow} GS[CS \longmapsto CS', NS \longmapsto NS'] \\ CS' = CS[x \longmapsto (\theta, \sigma)] \\ NS' = NS[B \longmapsto B \oplus (x, m, y)] \end{array}} \tag{17}$$

# 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?

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

Defining Concurrent Continuous behavior expiration detection