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.
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_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.

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.

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

2.2 Coarse-Grained Rules

Message Execution (FIFO)
(Labels must be retained for in execute)

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

Message Execution (Priority-based)

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

Continuous Behavior Expiration

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

CAN Bus

$$\frac{\begin{array}{c} NS = (B, i) \wedge i = true \wedge no\_software\_action(GS) \\ (x, m, y) = highest\_priority\_packet(B) \\ d = network\_delay(x, m, y) \\ ES = (events, p) \end{array}}{\begin{array}{c} GS \underset{n}{\rightarrow} GS[NS \longmapsto NS', ES \longmapsto ES'] \\ ES = ES[events \longmapsto events \oplus (d, \{send\_direct(y, m), i = true\})] \\ NS' = NS[B \longmapsto B \backslash (x, m, y), i \longmapsto false] \end{array}} \tag{4}$$

Events (How does the variable pool changes?)

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

2.3 Fine-Grained Rules

Go To

$$\frac{\begin{array}{c} DS(x) = (v, q, (p\ goto\ m')|\sigma) \\ CS(p) = m \end{array}}{\begin{array}{c} GS \overset{\tau}{\Rightarrow} GS[DS \longmapsto DS^{'}, CS \longmapsto CS^{'})] \\ DS^{'} = DS[x \longmapsto (c, q, \sigma)] \\ CS^{'} = CS[p \longmapsto m'] \end{array}} \tag{6}$$

Delay Statement

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

Continuous Variable Assignment

$$\frac{...}{GS \overset{cvar=eval(expr)}{\Rightarrow} GS'} \tag{8}$$

Discrete Variable Assignment

$$\frac{DS(x) = (v, q, (dvar = expr|\sigma))}{GS \overset{\tau}{\Rightarrow} GS[DS \longmapsto DS']} \quad DS^{'} = DS[x \longmapsto (v[dvar \longmapsto eval(expr)], q, \sigma)] \tag{9}$$

Conditional True

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

Conditional False

$$\frac{DS(x) = (v, q, (if\ expr\ \sigma\ else\ \sigma'|\sigma'')) \quad eval(expr) = False}{GS \overset{\tau}{\Rightarrow} GS[DS \longmapsto DS']} \quad DS^{'} = DS[x \longmapsto (v, q, \sigma' \oplus \sigma'')] \tag{11}$$

Direct Message Send (Software Actor)

$$\frac{DS(x) = (v_x, q_x, ((y, m)|\sigma_x)) \quad DS(y) = (v_y, q_y, \sigma_y) \quad is\_direct(x, y)}{GS \overset{\tau}{\Rightarrow} GS[DS \longmapsto DS']} \quad DS^{'} = DS[x \longmapsto (v_x, q_x, \sigma_x), y \longmapsto (v_y, q_y \oplus m, \sigma_y)] \tag{12}$$

CAN Message Send (Software Actor)

$$\frac{DS(x) = (v_x, q_x, ((y, m)|\sigma_x)) \quad NS = (B, i) \quad is\_nat(x, y)}{GS \overset{\tau}{\Rightarrow} GS[DS \longmapsto DS', NS \longmapsto NS']} \quad DS^{'} = DS[x \longmapsto (v_x, q_x, \sigma_x)] \quad NS' = NS[B \longmapsto B \oplus (x, y, n)] \tag{13}$$

Direct Message Send (Physical Actor)

$$\frac{\dots}{G \overset{\tau}{\Rightarrow} G[DS \longmapsto DS']} \qquad (14)$$
$$DS^{'} = DS[x \longmapsto (v_x, q_x, \sigma_x), y \longmapsto (v_y, q_y \oplus m, \sigma_y)]$$

CAN Message Send (Physical Actor)

$$\frac{\dots}{GS \overset{\tau}{\Rightarrow} GS[DS \longmapsto DS', NS \longmapsto NS']} \qquad (15)$$
$$NS' = NS[B \longmapsto B \oplus (x, m, y)]$$

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?

Defining Concurrent Continuous behavior expiration detection

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