lil Formalisation

Vincent Lecrubier

March 18, 2014

Writing conventions

We use the following writing conventions:

Variables:	$name$
Functions:	name
Anonymous functions:	$a \longmapsto b$
Tuple:	$\langle a, b, c\rangle$
Vector:	(a, b, c)
Set or alphabet:	$\{a, b, c\}$
Power set of set A:	$\mathcal{P}(A)$
Set of finite sequences of elements of set A:	A^*
Set of infinite sequences of elements of set A:	$A^{\mathbb{N}}$
Finite sequences:	$(a_1, a_2, ..., a_k) \in A^*$
Infinite sequences:	$(a_n)_{n\in\mathbb{N}} \in A^{\mathbb{N}}$
Indexed element of a vector:	$A[n] = a_n$
Sets of atomic elements:	$\mathbb{S}$
Sets of tuples:	$\mathbf{S}$

The function $\mathrm{index\_of}(vector, value)$ returns the first index of a value in a vector.
The function $\dim(vector)$ returns the dimension of a vector.

1 Formal model

1.1 General concepts

1.1.1 Value

A value v is a piece of data that lil models processes. The set of all possible values is called $\mathbb{V}$. For a first approach, we can simplify by restricting values to $\mathbb{N}$:

$$v \in \mathbb{V} = \mathbb{N}$$

1.1.2 Time stamp

A time stamp t represents the time interval between a reference date and another date. The set of all possible time stamps is called $\mathbb{T}$. For a first approach, we can simplify by restricting time stamps to $\mathbb{N}$:

$$t \in \mathbb{T} = \mathbb{N}$$

1.1.3 Identifier

An identifier is a word used to identify objects of a lil model. There are syntactic constraints on the words in order for them to be valid identifiers. The set of all possible identifiers is called $\mathbb{I}$. Identifiers will often be noted id.

$$id \in \mathbb{I}$$

There is a bijection between $\mathbb{I}$ and $\mathbb{N}$ and we will call it $\mathrm{id\_code}$:

$$\mathrm{id\_code}\colon \mathbb{I} \longleftrightarrow \mathbb{N}$$

We define the function $\mathrm{id\_index}$ which gives the index of a specific identifier in a vector of identifiers:

$$\mathrm{id\_index}\colon \left| \begin{array}{rcl} \mathbb{I} \times \mathbb{I}^* & \longrightarrow & \mathbb{N} \\ (id, (id_1, id_2, ..., id_n)) & \longmapsto & \begin{cases} \varnothing & \nexists i \in [\![1, n]\!], id_i = id \\ i \mid id_i = id & \exists i \in [\![1, n]\!], id_i = id \end{cases} \end{array} \right.$$

1.1.4 Timed value

Most of lil computations involve timed values. A timed value u is a couple made of a time stamp $t \in \mathbb{T}$ and a value $v \in \mathbb{V}$:

$$u = \langle t, v \rangle$$

The set of all possible timed values is called $\mathbf{U}$:

$$\mathbf{U} = \mathbb{T} \times \mathbb{V}$$

We define the function $\mathrm{u\_value}$ which gives the value of a timed value:

$$\mathrm{u\_value}\colon \left| \begin{array}{rcl} \mathbf{U} & \longrightarrow & \mathbb{V} \\ \langle t, v \rangle & \longmapsto & v \end{array} \right.$$

We define the function $\mathrm{u\_time}$ which gives the time stamp of a timed value:

$$\mathrm{u\_time}\colon \left| \begin{array}{rcl} \mathbf{U} & \longrightarrow & \mathbb{T} \\ \langle t, v \rangle & \longmapsto & t \end{array} \right.$$

1.2 Specification concepts

1.2.1 Behavior

Behaviors are the building blocks of lil interaction specification. A l, m, n–behavior b is a tuple containing an identifier $id \in \mathbb{I}$, a vector of input ports identifiers $I \in \mathbb{I}^l$, a vector of state variable identifiers $S \in \mathbb{I}^m$, a vector of output ports identifiers $O \in \mathbb{I}^n$, and a transition function $f\colon \mathbf{U}^m \times (I \times \mathbf{U}) \longrightarrow \mathbf{U}^m \times (O \times \mathbf{U})^*$:

$$b = \langle id, I, S, O, f \rangle$$

The set of all possible l, m, n–behaviors is called $\mathbf{B}_{l,m,n}$ and we have:

$$\mathbf{B}_{l,m,n} = \mathbb{I} \times \mathbb{I}^l \times \mathbb{I}^m \times \mathbb{I}^n \times (\mathbf{U}^m \times (\mathbb{I} \times \mathbf{U})^*)^{\mathbf{U}^m \times (\mathbb{I} \times \mathbf{U})}$$

The set of all possible behaviors is called $\mathbf{B}$ and we have:

$$\mathbf{B} = \bigsqcup_{(l,m,n) \in \mathbb{N}^3} \mathbf{B}_{l,m,n}$$

The transition function f takes for input the current state of the behavior, and a single input event on one of its ports, and gives as output the next state of the behavior, and a vector of output events to be propagated. Less formally, we could write:

$$f(\mathit{current\ state}, \mathit{input\ event}) = (\mathit{next\ state}, \mathit{output\ events})$$

We define the function $\mathrm{b\_identifier}$ which gives the identifier of a behavior:

$$\mathrm{b\_identifier}\colon \left| \begin{array}{rcl} \mathbf{B} & \longrightarrow & \mathbb{I} \\ \langle id, I, S, O, V, f \rangle & \longmapsto & id \end{array} \right.$$

We define the function $\mathrm{b\_inputs}$ which gives the vector of inputs of a behavior:

$$\mathrm{b\_inputs}\colon \left| \begin{array}{rcl} \mathbf{B} & \longrightarrow & \mathbb{I}^* \\ \langle id, I, S, O, V, f \rangle & \longmapsto & I \end{array} \right.$$

We define the function $\mathrm{b\_state\_variables}$ which gives the vector of state variables of a behavior:

$$\mathrm{b\_state\_variables}\colon \left| \begin{array}{rcl} \mathbf{B} & \longrightarrow & \mathbb{I}^* \\ \langle id, I, S, O, V, f \rangle & \longmapsto & S \end{array} \right.$$

We define the function $\mathrm{b\_outputs}$ which gives the vector of outputs of a behavior:

$$\mathrm{b\_outputs}\colon \left| \begin{array}{rcl} \mathbf{B} & \longrightarrow & \mathbb{I}^* \\ \langle id, I, S, O, V, f \rangle & \longmapsto & O \end{array} \right.$$

We define the function $\mathrm{b\_transition\_function}$ which gives the transition function of a behavior:

$$\mathrm{b\_transition\_function}\colon \left| \begin{array}{rcl} \mathbf{B} & \longrightarrow & (\mathbf{U}^m \times (\mathbb{I} \times \mathbf{U})^*)^{\mathbf{U}^m \times (\mathbb{I} \times \mathbf{U})} \\ \langle id, I, S, O, V, f \rangle & \longmapsto & f \end{array} \right.$$

1.2.2 Connexion

A connexion c is a quadruple of identifiers containing a source behavior identifier $b_{source} \in \mathbb{I}$, the identifier of an output port of the source behavior $p_{source} \in \mathbb{I}$, a destination behavior identifier $b_{destination} \in \mathbb{I}$, the identifier of an input port of the destination behavior $p_{destination} \in \mathbb{I}$:

$$c = \langle b_{source}, p_{source}, b_{destination}, p_{destination} \rangle$$

The set of all possible connexions is called $\mathbf{C}$ and we have:

$$\mathbf{C} = \mathbb{I}^4$$

We define the function $\mathrm{c\_source\_behavior}$ which gives the identifier of the source behavior of a connexion:

$$\mathrm{c\_source\_behavior}: \left| \begin{array}{rcl} \mathbf{C} & \longrightarrow & \mathbb{I} \\ \langle b_{source}, p_{source}, b_{destination}, p_{destination} \rangle & \longmapsto & b_{source} \end{array} \right.$$

We define the function $\mathrm{c\_source\_port}$ which gives the identifier of the source port of a connexion:

$$\mathrm{c\_source\_port}: \left| \begin{array}{rcl} \mathbf{C} & \longrightarrow & \mathbb{I} \\ \langle b_{source}, p_{source}, b_{destination}, p_{destination} \rangle & \longmapsto & p_{source} \end{array} \right.$$

We define the function $\mathrm{c\_destination\_behavior}$ which gives the identifier of the destination behavior of a connexion:

$$\mathrm{c\_destination\_behavior}: \left| \begin{array}{rcl} \mathbf{C} & \longrightarrow & \mathbb{I} \\ \langle b_{source}, p_{source}, b_{destination}, p_{destination} \rangle & \longmapsto & b_{destination} \end{array} \right.$$

We define the function $\mathrm{c\_destination\_port}$ which gives the identifier of the destination port of a connexion:

$$\mathrm{c\_destination\_port}: \left| \begin{array}{rcl} \mathbf{C} & \longrightarrow & \mathbb{I} \\ \langle b_{source}, p_{source}, b_{destination}, p_{destination} \rangle & \longmapsto & p_{destination} \end{array} \right.$$

1.2.3 Interactor

An interactor i is a tuple containing a set of behaviors $B \subset \mathbf{B}$, and a set of connexions $C \subset \mathbf{C}$:

$$i = \langle B, C \rangle$$

The set of all possible interactors is called $\mathbf{I}$ and we have:

$$\mathbf{I} = \mathcal{P}(\mathbf{B}) \times \mathcal{P}(\mathbf{C})$$

We define the function $\mathrm{i\_behaviors}$ which gives the set of behaviors of an interactor:

$$\mathrm{i\_behaviors}: \left| \begin{array}{rcl} \mathbf{I} & \longrightarrow & \mathcal{P}(\mathbf{B}) \\ \langle B, C \rangle & \longmapsto & B \end{array} \right.$$

We define the function $\mathrm{i\_connexions}$ which gives the set of connexions of an interactor:

$$\mathrm{i\_connexions}: \left| \begin{array}{rcl} \mathbf{I} & \longrightarrow & \mathcal{P}(\mathbf{C}) \\ \langle B, C \rangle & \longmapsto & C \end{array} \right.$$

We define the function $\mathrm{i\_is\_consistent}$ which checks if an interactor is defined in a consistent way, without checking anything about its execution:

$$\mathrm{i\_is\_consistent}: \left| \begin{array}{rcl} \mathbf{I} & \longrightarrow & \mathbb{B} \\ \langle B, C \rangle & \longmapsto & \begin{array}{c} (\forall b_1 \in B, \forall b_2 \in B \setminus \{b1\}, \mathrm{b\_identifier}(b_1) \neq \mathrm{b\_identifier}(b_2)) \wedge \\ \left(\forall c \in C, \begin{array}{cc} \exists! b_s \in B & \begin{array}{c} (\mathrm{c\_source\_behavior}(c) = \mathrm{b\_identifier}(b_s)) \wedge \\ (\exists i \in \mathbb{N}, \mathrm{c\_source\_port}(c) = \mathrm{b\_outputs}(b_s)[i]) \end{array} \\ \exists! b_d \in B & \begin{array}{c} (\mathrm{c\_destination\_behavior}(c) = \mathrm{b\_identifier}(b_d)) \wedge \\ (\exists i \in \mathbb{N}, \mathrm{c\_destination\_port}(c) = \mathrm{b\_inputs}(b_d)[i]) \end{array} \end{array} \right) \end{array} \end{array} \right.$$

1.3 Run time concepts

1.3.1 State variable valuation

A state variable valuation w is a triple containing a behavior identifier $b \in \mathbb{I}$, a state variable identifier $s \in \mathbb{I}$, and a timed value $u \in \mathbf{U}$:

$$w = \langle b, s, u \rangle$$

The set of all possible state variables valuation is called $\mathbf{W}$ and we have:

$$\mathbf{W} = \mathbb{I} \times \mathbb{I} \times \mathbf{U}$$

We define the function $\mathrm{w\_behavior}$ which gives the identifier of the behavior whose state variable is being valued by a valuation:

$$\mathrm{w\_behavior}\colon \left| \begin{array}{rcl} \mathbf{W} & \longrightarrow & \mathbb{I} \\ \langle b, s, u \rangle & \longmapsto & b \end{array} \right.$$

We define the function $\mathrm{w\_variable}$ which gives the identifier of the state variable being valued by a valuation:

$$\mathrm{w\_variable}\colon \left| \begin{array}{rcl} \mathbf{W} & \longrightarrow & \mathbb{I} \\ \langle b, s, u \rangle & \longmapsto & s \end{array} \right.$$

We define the function $\mathrm{w\_value}$ which gives the timed value of the state variable associated with a valuation:

$$\mathrm{w\_value}\colon \left| \begin{array}{rcl} \mathbf{W} & \longrightarrow & \mathbb{V} \\ \langle b, s, u \rangle & \longmapsto & u \end{array} \right.$$

1.3.2 Interactor state

An interactor state q is a set of state variable valuations:

$$q = \{w_1, w_2, ..., w_n\}$$

The set of all possible interactor states is called $\mathbf{Q}$ and we have:

$$\mathbf{Q} = \mathbf{W}^*$$

We define the function $\mathrm{q\_is\_consistent}$ which checks if an interactor state is consistent with an interactor, *i.e.* whether an interactor state object correctly represents a state of a given interactor:

$$\mathrm{q\_is\_consistent}\colon \left| \begin{array}{rcl} \mathbf{Q} \times \mathbf{I} & \longrightarrow & \mathbb{B} \\ (q, i) & \longmapsto & \begin{array}{c} \forall w \in q, \exists! b \in \mathrm{i\_behaviors}(i), \left(\begin{array}{rcl} \mathrm{w\_behavior}(w) & = & \mathrm{b\_identifier}(b) \wedge \\ \mathrm{w\_variable}(w) & \in & \mathrm{b\_state\_variables}(b) \end{array} \right) \\ \wedge \\ \forall b \in \mathrm{i\_behaviors}(i), \exists! w \in q, \left(\begin{array}{rcl} \mathrm{w\_behavior}(w) & = & \mathrm{b\_identifier}(b) \wedge \\ \mathrm{w\_variable}(w) & \in & \mathrm{b\_state\_variables}(b) \end{array} \right) \end{array} \end{array} \right.$$

We can then define the set $\mathbf{Q}_i$ which contains all valid states of an interactor i, we will see later that $\mathbf{Q}_i$ can also be called the state set of i:

$$\mathbf{Q}_i = \{q \in \mathbf{Q} \mid \mathrm{q\_is\_consistent}(q, i)\}$$

1.3.3 Token

A token t is a couple containing a connexion $c \in \mathbf{C}$ and a timed value $u \in \mathbf{U}$:

$$t = \langle c, u \rangle$$

The set of all possible tokens is called $\mathbf{T}$ and we have:

$$\mathbf{T} = \mathbf{C} \times \mathbf{U}$$

We define the function $\mathrm{t\_connexion}$ which gives the connexion associated with a token:

$$\mathrm{t\_connexion}\colon \left| \begin{array}{rcl} \mathbf{T} & \longrightarrow & \mathbf{C} \\ \langle c, u \rangle & \longmapsto & c \end{array} \right.$$

We define the function $\mathrm{t\_value}$ which gives the timed value associated with a token:

$$\mathrm{t\_value}\colon \left| \begin{array}{ccc} \mathbf{T} & \longrightarrow & \mathbf{U} \\ \langle c, u\rangle & \longmapsto & u \end{array} \right.$$

We define the function $\mathrm{t\_is\_consistent}$ which checks if a token is consistent with an interactor, *i.e.* whether a token can be associated with a connexion within a given interactor:

$$\mathrm{t\_is\_consistent}\colon \left| \begin{array}{ccl} \mathbf{T}\times\mathbf{I} & \longrightarrow & \mathbb{B} \\ (t, i) & \longmapsto & \mathrm{t\_connexion}(t) \in \mathrm{i\_connexions}(i) \end{array} \right.$$

We can then define the set $\mathbf{T}_i$ which contains all valid tokens for an interactor i, we will see later that $\mathbf{T}_i$ can also be called the stack alphabet of i:

$$\mathbf{T}_i = \{t \in \mathbf{T} \mid \mathrm{t\_is\_consistent}(t, i)\}$$

1.3.4 Action

An action a consists of a behavior identifier $b \in \mathbb{I}$ and a timed value $u \in \mathbf{U}$:

$$a = \langle b, u\rangle$$

The set of all possible actions is $\mathbf{A}$ and we have:

$$\mathbf{A} = \mathbb{I} \times \mathbf{U}$$

We define the function $\mathrm{a\_actor}$ which gives the identifier of the behavior associated with an action. This behavior can also be called actor:

$$\mathrm{a\_actor}\colon \left| \begin{array}{ccc} \mathbf{A} & \longrightarrow & \mathbb{I} \\ \langle b, u\rangle & \longmapsto & b \end{array} \right.$$

We define the function $\mathrm{a\_value}$ which gives the timed value associated with an action:

$$\mathrm{a\_value}\colon \left| \begin{array}{ccc} \mathbf{A} & \longrightarrow & \mathbf{U} \\ \langle b, u\rangle & \longmapsto & u \end{array} \right.$$

We define the function $\mathrm{a\_is\_consistent}$ which checks if an action is consistent with an interactor, *i.e.* whether an action can be associated with a behavior, or actor, of a given interactor:

$$\mathrm{a\_is\_consistent}\colon \left| \begin{array}{ccl} \mathbf{A}\times\mathbf{I} & \longrightarrow & \mathbb{B} \\ (a, i) & \longmapsto & \exists! b \in \mathrm{i\_behaviors}(i), \mathrm{b\_identifier}(b) = \mathrm{a\_actor}(a) \end{array} \right.$$

We can then define the set $\mathbf{A}_i$ which contains all valid actions for an interactor i, we will see later that $\mathbf{A}_i$ can also be called the input alphabet of i:

$$\mathbf{A}_i = \{a \in \mathbf{A} \mid \mathrm{a\_is\_consistent}(a, i)\}$$

1.3.5 Interactor transition function

The transition function δ_i associated with an interactor i is the following application:

$$\delta_i\colon \left| \begin{array}{rcl} \mathbf{Q}_i \times (\mathbf{A}_i \cup \{\varepsilon\}) \times (\mathbf{T}_i \cup \{\tau\}) & \longrightarrow & \mathbf{Q}_i \times \mathbf{T}_i^* \\ (q, a, t) & \longmapsto & \textsc{transition\_function}(i, q, a, t) \end{array} \right.$$

Note that ε is the empty action and τ is the end of stack symbol.

Algorithm 1 Transition function of interactor i in state q, receiving an action a, while token t is on top of the stack

Require: $\mathrm{q\_is\_consistent}(q, i)$

function TRANSITION_FUNCTION(i, q, a, t)
  if $(a \neq \varepsilon) \wedge (t = \tau)$ **then** ▷ No token left, but an action incoming
    return PROCESS_ACTION(i, q, a) ▷ Process the action: no state change, but the stack grew
  else if $(a = \varepsilon) \wedge (t \neq \tau)$ **then** ▷ Token left in the stack, no action to process
    return PROCESS_TOKEN(i, q, t) ▷ Process the token: state changed, and the stack grew
  else ▷ Since this is a determinist stack automata
    return "undefined" ▷ We can't process an action and a token at the same time
  end if
end function

Algorithm 2 Process action a when interactor i is in state q

Require: $\text{q\_is\_consistent}(q, i) \wedge a \neq \varepsilon$

function PROCESS_ACTION(i, q, a)
    $new\_tokens \leftarrow ()$ ▷ So far, no new tokens to be added to the stack
    for all $c \in \text{i\_connexions}(i)$ **do** ▷ Check all connexions of the interactor
        if $\text{c\_source\_behavior}(c) = \text{a\_actor}(a) \wedge \text{c\_source\_port}(c) =$“out” **then** ▷ A matching connexion !
            $new\_tokens \leftarrow \langle c, \text{a\_value}(a)\rangle; new\_tokens$ ▷ Add a new token for the matching connexion
        end if ▷ Nothing to do if the connexion does not match
    end for ▷ Finished scanning connexions
    return $\langle q, new\_tokens\rangle$ ▷ An action was processed: no state change, but the stack grew
end function

Algorithm 3 Process token t when interactor i is in state q

Require: $\text{q\_is\_consistent}(q, i) \wedge t \neq \tau$

Ensure: $\text{q\_is\_consistent}(new\_q, i)$

function PROCESS_TOKEN(i, q, t)
    $b \leftarrow$ GET_BEHAVIOR$(i, \text{c_destination_behavior}(\text{t_connexion}(t)))$ ▷ Get destination behavior
    $b\_state \leftarrow$ GET_BEHAVIOR_STATE(b, q) ▷ Extract the destination behavior state vector
    $message\_to\_b \leftarrow$ TOKEN_TO_MESSAGE(t) ▷ Adapt the token so the transition function can use it
    $\langle b\_state\_new, messages\_from\_b\rangle \leftarrow \text{b\_transition\_function}(b)(b\_state, message\_to\_b)$ ▷ Transition function
    $new\_t \leftarrow$ TOKENS_FROM_MESSAGES$(i, b, messages_from_b)$
    $new\_q \leftarrow$ SET_BEHAVIOR_STATE(b, q, b_state_new)
    return $\langle new\_q, new\_t\rangle$ ▷ A token was processed: state changed, and the stack grew
end function

Algorithm 4 Get the behavior with identifier id within interactor i

Require: $\text{i\_is\_consistent}(i)$

function GET_BEHAVIOR(i, id)
    if $\exists b \in \text{i\_behaviors}(i), \text{b\_identifier}(b) = id$ **then**
        return b ▷ There should be only one possible b since i is consistent.
    else
        return $null$ ▷ Not found.
    end if
end function

Algorithm 5 Transform a token t into a message sent to the behavior transition function

function TOKEN_TO_MESSAGE(t)
    return $\langle \text{c\_destination\_port}(\text{t\_connexion}(t)), \text{t\_value}(t)\rangle$
end function

Algorithm 6 Transform messages m sent by the transition function of behavior b into tokens for interactor i

function TOKENS_FROM_MESSAGES(i, b, M)
    $result \leftarrow ()$
    for all $m \in M$ **do** ▷ For all messages sent by behavior
        for all $c \in \text{i\_connexions}(i)$ **do** ▷ For all connexions linked to the message's port
            if $\text{c\_source\_behavior}(c) = \text{b\_identifier}(b) \wedge \text{c\_source\_port}(c) = m[1]$ **then**
                $result \leftarrow \langle c, m[2]\rangle; result$ ▷ Add a new token to the stack
            end if
        end for
    end for
    return $result$
end function

Algorithm 7 Get the state vector of behavior b from state q of its containing interactor

Require: $\exists i \in \mathbf{I}, \text{q\_is\_consistent}(q, i) \wedge b \in \text{i\_behaviors}(i)$
Ensure: $\dim(result) = \dim(\text{b\_state\_variables}(b)) \wedge \forall k \in [\![1, \dim(result)]\!], result[k] \neq \bot$

function GET_BEHAVIOR_STATE(b, q)
  $result \leftarrow (\bot, ..., \bot)$
  for all $w \in q$ **do**
    if $\text{w\_behavior}(w) = \text{b\_identifier}(b)$ **then**
      if $\exists k \in [\![1, \dim(result)]\!], \text{b\_state\_variables}(b)[k] = \text{w\_variable}(w)$ **then**
        $result[k] \leftarrow \text{w\_value}(w)$
      end if
    end if
  end for
  return $result$ ▷ Result will not contain any $\bot$ since q is consistent with the interactor i containing b
end function

Algorithm 8 Change an existing interactor state q so that the state vector of behavior b becomes s

Require: $(\exists i \in \mathbf{I}, \text{q\_is\_consistent}(q, i) \wedge b \in \text{i\_behaviors}(i)) \wedge \dim(s) = \dim(\text{b\_state\_variables}(b))$
Ensure: $\exists i \in \mathbf{I}, \text{q\_is\_consistent}(q, i) \wedge b \in \text{i\_behaviors}(i)$

function SET_BEHAVIOR_STATE(b, q, s)
  $result \leftarrow q$ ▷ We start by taking the current state
  for all $w \in q$ **do**
    if $\text{w\_behavior}(w) = \text{b\_identifier}(b)$ **then**
      if $\exists k \in [\![1, \dim(s)]\!], \text{b\_state\_variables}(b)[k] = \text{w\_variable}(w)$ **then**
        $new\_w \leftarrow \langle \text{w\_behavior}(w), \text{w\_variable}(w), s[k] \rangle$ ▷ Only change the value of the valuation
        $result \leftarrow (result \setminus \{w\}) \cup \{new\_w\}$ ▷ Replace old valuation by new one
      end if
    end if
  end for
  return $result$ ▷ Result is consistent because q is consistent and we did not change its structure
end function

1.3.6 Interactor execution

From any interactor i we can define a pushdown automaton m and we have:

$$M_i = \langle \mathbf{Q}_i, \mathbf{A}_i, \mathbf{T}_i, \delta_i, q_i^0, \tau, \mathbf{Q}_i \rangle$$

The execution of this pushdown automaton is the execution of the interactor.

2 Language semantics

2.1 Interpretation

2.1.1 Starting point

The empty interactor has no behavior and no connexion:

$$i_0 = \langle \varnothing, \varnothing \rangle$$

The starting point for interpretation of the input program X is the empty interactor i_0, and the empty scope $\varnothing$:

$$< X \mid i_0, \varnothing >$$

The scope is the identifier of the behavior which syntactically owns the words being interpreted...

2.1.2 Interpretation of language constructs

Actor An actor is a simple behavior that is used as a bootstrap to inject tokens from external actions.

$$\text{B\_actor}(id) = \langle \text{“actor”} + id, (\text{“in”}), (), (\text{“out”}), () \longmapsto () \rangle$$

The interpretation of the actor construct adds one behavior to the interactor:

$$< \mathit{name}\texttt{: actor;} X \mid \langle B, C \rangle, \mathit{scope} > \longrightarrow < X \mid \langle B \cup \{\text{B\_actor}(\mathit{name})\}, C \rangle, \mathit{name} >$$

Event An event is a simple behavior with one input and one output that forwards all tokens it receive.

$$\text{B\_event}(id) = \left\langle \text{“event”} + id, (\text{“in”}), (), (\text{“out”}), ((\alpha), \langle \rho, \nu \rangle) \longmapsto \begin{cases} ((\nu), (\langle \text{“out”}, \nu \rangle)) & \rho = \text{“in”} \\ ((\alpha), ()) & \rho \neq \text{“in”} \end{cases} \right\rangle$$

The interpretation of the event construct adds one behavior to the interactor:

$$< \mathit{name}\texttt{: event;} X \mid \langle B, C \rangle, \mathit{scope} > \longrightarrow < X \mid \langle B \cup \{\text{B\_event}(\mathit{name})\}, C \rangle, \mathit{name} >$$

The “from” and “to” clauses respectively indicate that an event receives and send events to a specified actor, so they add the required connexions:

$$< \texttt{from}\ \mathit{name}; X \mid \langle B, C \rangle, \mathit{scope} > \longrightarrow < X \mid \langle B, C \cup \{\langle \mathit{name}, \text{“out”}, \mathit{scope}, \text{“in”} \rangle\} \rangle, \mathit{scope} >$$

$$< \texttt{to}\ \mathit{name}; X \mid \langle B, C \rangle, \mathit{scope} > \longrightarrow < X \mid \langle B, C \cup \{\langle \mathit{scope}, \text{“out”}, \mathit{name}, \text{“in”} \rangle\} \rangle, \mathit{scope} >$$

Flow A flow is similar to an event, except that it has one state variable representing the last received value, and only forwards token if their value is different than the last one received:

$$\begin{aligned} \text{B\_flow}(id) \quad = \quad & \Big\langle \text{“flow”} + id, (\text{“in”}), (\text{“last\_value”}), (\text{“out”}), \\ & ((\alpha), \langle \rho, \nu \rangle) \longmapsto \begin{cases} ((\nu), (\langle \text{“out”}, \nu \rangle)) & \rho = \text{“in”} \wedge \alpha \neq \nu \\ ((\alpha), ()) & \rho = \text{“in”} \wedge \alpha = \nu \\ ((\alpha), ()) & \rho \neq \text{“in”} \end{cases} \Big\rangle \end{aligned}$$

The interpretation of the flow construct adds one behavior to the interactor:

$$< \mathit{name}\texttt{: flow;} X \mid \langle B, C \rangle, \mathit{scope} > \longrightarrow < X \mid \langle B \cup \{\text{B\_flow}(\mathit{name})\}, C \rangle >, \mathit{name} >$$

The “from” and “to” clauses respectively indicate that a flow receives and send flows to a specified actor, so they add the required connexions:

$$< \texttt{from}\ \mathit{name}; X \mid \langle B, C \rangle, \mathit{scope} > \longrightarrow < X \mid \langle B, C \cup \{\langle \mathit{name}, \text{“out”}, \mathit{scope}, \text{“in”} \rangle\} \rangle, \mathit{scope} >$$

$$< \texttt{to}\ \mathit{name}; X \mid \langle B, C \rangle, \mathit{scope} > \longrightarrow < X \mid \langle B, C \cup \{\langle \mathit{scope}, \text{“out”}, \mathit{name}, \text{“in”} \rangle\} \rangle, \mathit{scope} >$$

Expression An expression is a behavior that stores the last values of its inputs, and outputs the result of its evaluation every time it receives a token:

$$\text{B\_expr}(expr) \quad = \quad \Bigg\langle \text{“expr”} + expr, (\text{“in1”}, \text{“in2”}, ..., \text{“ink”}), (\text{“last\_in1”}, \text{“last\_in2”}, ..., \text{“last\_ink”}), (\text{“out”}),$$

$$((\alpha_1, ..., \alpha_k), \langle \rho, \nu \rangle) \longmapsto \begin{cases} ((\nu, \alpha_2, ..., \alpha_k), (\langle \text{“out”}, \text{eval}(expr, (\nu, \alpha_2, ..., \alpha_k)) \rangle)) & \rho = \text{“in1”} \\ ((\alpha_1, \nu, ..., \alpha_k), (\langle \text{“out”}, \text{eval}(expr, (\alpha_1, \nu, ..., \alpha_k)) \rangle)) & \rho = \text{“in2”} \\ \vdots & \vdots \\ ((\alpha_1, \alpha_2, ..., \nu), (\langle \text{“out”}, \text{eval}(expr, (\alpha_1, \alpha_2, ..., \nu)) \rangle)) & \rho = \text{“ink”} \end{cases} \Bigg\rangle$$

The interpretation of expressions constructs adds one behavior to the interactor, and adds one connexion from this behavior output to the scope input:

$$< \texttt{expr(a1,a2,..,ak);} X \mid \langle B, C \rangle, scope > \longrightarrow$$
$$< X \mid \langle B \cup \{\text{B\_expr}(expr)\}, C \cup \{\langle \text{b\_identifier}(\text{B\_expr}(expr)), \text{“out”}, scope, \text{“in”} \rangle\}\rangle, scope >$$

Assignment An assignment is a behavior that forwards values it receives on its value port, continuously while it is active, or punctually when asked through its trigger input:

$$\text{B\_assign}(idd, ids) \quad = \quad \Bigg\langle \text{“assign”} + idd + ids, (\text{“in”}, \text{“activation”}, \text{“trigger”}), (\text{“last\_in”}, \text{“active”}), (\text{“out”}),$$

$$((\alpha_1, \alpha_2), \langle \rho, \nu \rangle) \longmapsto \begin{cases} ((\nu, \alpha_2), (\langle \text{“out”}, \nu \rangle)) & \rho = \text{“in”} \wedge \alpha_2 \\ ((\nu, \alpha_2), ()) & \rho = \text{“in”} \wedge \neg \alpha_2 \\ ((\alpha_1, \nu), (\langle \text{“out”}, \alpha_1 \rangle)) & \rho = \text{“activation”} \wedge \nu \\ ((\alpha_1, \nu), ()) & \rho = \text{“activation”} \wedge \neg \nu \\ ((\alpha_1, \alpha_2), (\langle \text{“out”}, \alpha_1 \rangle)) & \rho = \text{“trigger”} \\ ((\alpha_1, \alpha_2), ()) & otherwise \end{cases} \Bigg\rangle$$

The interpretation of the assign construct adds one behavior to the interactor, and adds two connexions: one from the assignment behavior to its destination (e.g. a flow) and another one to the behavior from its source (e.g. an expression):

$$\begin{aligned} &< left \ = \ right; X \mid \langle B, C \rangle, scope > \longrightarrow \\ &\quad < X \mid \langle B \cup \{\text{B\_assign}(left, right)\}, \\ &\quad\quad C \cup \{ \\ &\quad\quad\quad \langle \text{b\_identifier}(\text{B\_assign}(left, right)), \text{“out”}, left, \text{“in”} \rangle, \\ &\quad\quad\quad \langle right, \text{“out”}, \text{b\_identifier}(\text{B\_assign}(left, right)), \text{“in”} \rangle \\ &\quad \}\rangle, scope > \end{aligned}$$

On A on behavior triggers behaviors that are in its scope. Ons can be activated or deactivated. Activation or deactivation of a on behavior does not trigger behaviors that are in its scope.

$$\text{B\_on}(event) \quad = \quad \Bigg\langle \text{“on”} + event, (\text{“in”}, \text{“activation”}), (\text{“active”}), (\text{“out”}),$$

$$((\alpha), \langle \rho, \nu \rangle) \longmapsto \begin{cases} ((\nu), ()) & \rho = \text{“activation”} \\ ((\alpha), (\langle \text{“out”}, T \rangle)) & \rho = \text{“in”} \wedge \alpha \\ ((\alpha), ()) & \rho = \text{“in”} \wedge \neg \alpha \\ ((\alpha), ()) & otherwise \end{cases} \Bigg\rangle$$

The interpretation of the on construct adds one behavior to the interactor, and adds connexions from the output port of this behavior to the trigger ports of behaviors in its scope:

$$
\begin{aligned}
&< \texttt{on}\ a\texttt{:}\ b; X \mid \langle B, C\rangle, \mathit{scope} > \longrightarrow \\
&\quad < X \mid \langle B \cup \{\text{B\_on}(a)\}, \\
&\qquad C \cup \{ \\
&\qquad\quad \langle \text{b\_identifier}(a), \text{“out”}, \text{b\_identifier}(\text{B\_on}(a)), \text{“in”}\rangle, \\
&\qquad\quad \langle \text{b\_identifier}(\text{B\_on}(a)), \text{“out”}, \text{b\_identifier}(b), \text{“trigger”}\rangle \\
&\quad \}\rangle, \mathit{scope} >
\end{aligned}
$$

When A when behavior activates or deactivates behaviors that are in its scope. Whens can be activated or deactivated, so they are nestable. Activation or deactivation of a when behavior does trigger the activation or deactivation of behaviors that are in its scope.

$$
\text{B\_when}(\mathit{cond}) = \left\langle \begin{array}{l} \text{“when”} + \mathit{cond}, (\text{“in”}, \text{“activation”}), (\text{“active”}), (\text{“out”}), \\ ((\alpha), \langle \rho, \nu\rangle) \longmapsto \begin{cases} ((\nu), (\langle \text{“out”}, T\rangle)) & \rho = \text{“activation”} \wedge \nu \\ ((\nu), (\langle \text{“out”}, F\rangle)) & \rho = \text{“activation”} \wedge \neg\nu \\ ((\nu), (\langle \text{“out”}, T\rangle)) & \rho = \text{“in”} \wedge \nu \\ ((\nu), (\langle \text{“out”}, F\rangle)) & \rho = \text{“in”} \wedge \neg\nu \\ ((\alpha), ()) & \mathit{otherwise} \end{cases} \end{array} \right\rangle
$$

The interpretation of the on construct adds one behavior to the interactor, and adds connexions from the output port of this behavior to the activation ports of behaviors in its scope:

$$
\begin{aligned}
&< \texttt{when}\ a\texttt{:}\ b; X \mid \langle B, C\rangle, \mathit{scope} > \longrightarrow \\
&\quad < X \mid \langle B \cup \{\text{B\_when}(a)\}, \\
&\qquad C \cup \{ \\
&\qquad\quad \langle \text{b\_identifier}(a), \text{“out”}, \text{b\_identifier}(\text{B\_when}(a)), \text{“in”}\rangle, \\
&\qquad\quad \langle \text{b\_identifier}(\text{B\_when}(a)), \text{“out”}, \text{b\_identifier}(b), \text{“activation”}\rangle \\
&\quad \}\rangle, \mathit{scope} >
\end{aligned}
$$

3 Case study

```
mode data:
   symbol in {"Off","Limit","Control"}

test interactor:
   driver : human actor
   car : system actor

   step : number constant
   minTarget : number constant
   maxTarget : number constant

   topLine : text or number flow to driver
   bottomLine : text or number flow to driver

   increment : void event from driver
   decrement : void event from driver

   targetSpeed : number flow to car
   actualSpeed : number flow from car to driver

   toggle : boolean flow

   switch : void event from driver
   modeDriver : mode flow from driver

   modeCar : mode flow to car

   alert : boolean flow to driver

   throttle : number flow from driver
   clutch : number flow from driver
   brake : number flow from driver

   topLine = if modeCar == "Off" then "128920 km" else modeCar
   bottomLine = if modeCar == "Off" then "1234 km" else targetSpeed

   on increment : targetSpeed = targetSpeed + step
               toggle = true
   on decrement : targetSpeed = targetSpeed - step
               toggle = true

   targetSpeed = crop(minTarget,maxTarget,if toggle then targetSpeed else round(
      actualSpeed,step))

   toggle = clutch < 0.01 && brake < 0.01 && toggle && modeDriver!="Off"

   on switch : toggle = !toggle

   modeCar = if toggle then modeDriver else "Off"

   alert = actualSpeed > targetSpeed && modeCar!="Off"
```