IF-2.0 Common Language Operational Semantics

Marius Bozga Yassine Lakhnech

VERIMAG, Centre Equation, 2 avenue de Vignate, F-38610 Gières

September 29, 2002

Abstract

This report describes the operational semantics of the common language IF-2.0. This semantics is taken as a reference point for the integration of tools. The key issue in defining the semantics concerns the smooth combination of dynamic process creation, handling process identifiers (pids), real-time aspects as well as asynchronous communication via timed-fifo queues.

1 Abstract Syntax

We deal with systems composed from active objects (process instances) which are executed in parallel and interact asynchronously through message (signal instances) passing.

Definition 1.1 (system) $(\mathcal{D}, \mathcal{S}, \mathcal{P})$ where

- $\mathcal{D}$ is a set of *data types* including *boolean*, *integer*, *clock*, and *pid*
- $\mathcal{S}$ is a set of *typed signals*
- $\mathcal{P}$ is a set of *process types*

Processes are defined as extended finite state-machines. Control states are partitioned between two disjoint classes: Q^+ stable states and Q^- unstable states.

Definition 1.2 (process) $(Q, X, T, q_0) \in \mathcal{P}$ where

- $Q = Q^+ \cup Q^-$ is a set of *control states* (stable and unstable)
- X is a set of *typed variables*
- T is a set of *transitions*
- $q_0 \in Q^+$ is the *initial state*

As usual, each transition has an enabling condition (guard) and an action. The urgency attribute denotes its priority with respect to the progress of time.

Definition 1.3 (transition) $q \xrightarrow[u]{g\ a} q' \in T$ where

- $q \in Q$ is the *source* state
- $g \in Exp(X)$ is a boolean *guard* condition
- a is an *action*

- $u \in \{eager, lazy\}$ is an *urgency type*
- $q' \in Q$ is the *target* state

Five categories of actions are considered : assignements, signal inputs and outputs, process creation and destruction. The abstract syntax is given below.

Definition 1.4 (action) An action $a \in Act(P)$ could be

- $x := e$ *assignment* of $e \in Exp(X)$ to $x \in X$
- $?s(x)$ *input* of signal $s \in \mathcal{S}$ with parameter $x \in X$
- $e_p!s(e)$ *output* of signal $s \in \mathcal{S}$ with parameter $e \in Exp(X)$ to instance $e_p \in Exp(X)$
- $x_p :=\uparrow P$ *create* new instance of process $P \in \mathcal{P}$ and get the fresh allocated pid in variable $x_p \in X$
- $\downarrow e_p$ *destroy* instance $e_p \in Exp(X)$.

The abstract syntax cover most of the IF-2.0 features. High-level constructs can be explicitely removed, for example:

- *delayable transitions*: they could be replaced by one lazy plus one eager transition;
- *structured control*: recursively, compound states are eliminated by adding outgoing transitions to their components and also, structured transitions are splitted using as much unstable states as needed;
- *internal signalroutes*: they are just normal processes with some predefined behaviour, depending on parameters such as reliability, queueing and delivery policy, delays, etc;

2 Operational Semantics: Configurations

We consider the standard domains $\mathbb{B}$ of booleans, $\mathbb{Z}$ of integers, $\mathbb{R}$ of reals with their usual interpretation. In addition, we consider some countable, infinite set $\mathbb{P}$ of process identifiers.

Definition 2.1 (data domains)

- $\mathbb{B}$ *boolean* domain
- $\mathbb{Z}$ *integer* domain
- $\mathbb{R}$ *real* domain
- $\mathbb{P}$ *pid* domain (countable, infinite)
- $\mathbb{D}$ *universal* (all data) domain

Messages are pairs (signal, value) which are well-typed i.e, the value is in the domain of the type of the signal.

Definition 2.2 (message domain)

- $\mathbb{M} \subseteq \mathcal{S} \times \mathbb{D}$ well-typed, $(s, d) \in \mathbb{D}$ noted as $s(d)$

Process configurations (process instances) are tuples including the process identifier, the control state, the queue of pending signals, and the (well-typed) variable context. In particular, clocks values are real numbers.

Definition 2.3 (process configuration) $\pi = (k, p, w, v)$

- $k \in \mathbb{P}$ *self* process instance identifier
- $p \in Q$ local *control* state
- $w \in \mathbb{M}^*$ *queue context*
- $v : X \to \mathbb{D}$ well-typed *variable context*

Now, system configurations(system instances) are sets of process configurations (process instances) having all distinct process identifiers (the well-formed condition below).

Definition 2.4 (system configuration) $\sigma = \{\pi_1, \pi_2, ...\pi_n, ...\}$

- well-formed condition: $\forall i, j.\ i \neq j \Rightarrow \pi_i.k \neq \pi_j.k$

A configuration is stable if *all* process instances are in stable states

Definition 2.5 (stable configuration) $\sigma = \{\pi_1, \pi_2, ...\pi_n, ...\}$ *stable*

- stability condition: $\forall i.\pi_i.q \in Q^+$

3 Operational Semantics: Moves

For sake of clarity, we consider first the *local semantics* i.e, *local conditions* required in order to execute process transitions and *global transformations* induced by them on configurations. For each action type (assignment, signal input and output, process creation and destruction) we have one semantic rule.

Definition 3.1 (local semantics)

- internal computation, assignement

$$\boxed{\dfrac{q \xrightarrow[u]{g\ x := e} q' \quad \begin{array}{l} \pi.p = q \\ \pi.v(g) = t \\ \pi.v(e) = d \end{array}}{\{\pi\} \xrightarrow[u]{\tau} \{\pi[\begin{array}{l} p \mapsto q' \\ v \mapsto v[x \mapsto d] \end{array}]\}}}$$

- inter process communication: signal input and output

$$\boxed{\dfrac{q \xrightarrow[u]{g\ ?s(x)} q' \quad \begin{array}{l} \pi.p = q \\ \pi.v(g) = t \\ \pi.w = s(d).z \end{array}}{\{\pi\} \xrightarrow[u]{?s(d)} \{\pi[\begin{array}{l} p \mapsto q' \\ w \mapsto z \\ v \mapsto v[x \mapsto d] \end{array}]\}}} \qquad \boxed{\dfrac{q \xrightarrow[u]{g\ e_p!s(e)} q' \quad \begin{array}{l} \pi.p = q \\ \pi.v(g) = t \\ \pi.v(e) = d \\ \pi.v(e_p) = k_0 \end{array} \quad \begin{array}{l} \pi'.k = k_0 \\ \pi'.w = z \end{array}}{\{\pi, \pi'\} \xrightarrow[u]{!s(d)} \{\pi[p \mapsto q'], \pi'[w \mapsto z.s(d)]\}}}$$

- process creation and destruction

$$\frac{q \xrightarrow[u]{g\ x := \uparrow P} q' \quad \begin{array}{l}\pi.p = q \\ \pi.v(g) = t\end{array} \quad \begin{array}{l}\pi'.k = k_0 \\ \pi'.p = P.q_0 \\ \pi'.v = P.v_0 \\ \pi'.w = \epsilon\end{array} \quad k_0 = \texttt{new}}{\{\pi\} \xrightarrow[u]{\uparrow P} \{\pi[\begin{array}{l}p \mapsto q' \\ v \mapsto v[x \mapsto k_0]\end{array}], \pi'\}}$$

$$\frac{q \xrightarrow[u]{g\ \downarrow e_p} q' \quad \begin{array}{l}\pi.p = q \\ \pi.v(g) = t \\ \pi.v(e_p) = k_0\end{array} \quad \pi'.k = k_0}{\{\pi, \pi'\} \xrightarrow[u]{\downarrow} \{\pi[p \mapsto q']\}}$$

We define now the *global semantics* i.e, the global conditions required for process transitions to be executed as well as time progress rules. On one hand, transitions issued from instances in unstable states have greater priority. On the other hand, time progress only in stable configurations as long as where are no eager transitions enabled.

Definition 3.2 (global semantics)

- unstable vs. stable transitions

$$\frac{\neg\text{stable}(\sigma') \quad \sigma' \xrightarrow[u]{l} \sigma''}{\sigma \cup \sigma' \xrightarrow[u]{l} \sigma \cup \sigma''} \qquad \frac{\text{stable}(\sigma) \quad \sigma' \xrightarrow[u]{l} \sigma''}{\sigma \cup \sigma' \xrightarrow[u]{l} \sigma \cup \sigma''}$$

- time semantics

$$\frac{\begin{array}{c}\delta \in \mathbb{R}^+ \quad \text{stable}(\sigma) \\ \{\sigma' \xrightarrow[\text{eager}]{l} \sigma'' \mid \exists \delta' \in [0, \delta).\sigma' = \sigma \oplus \delta'\} = \emptyset\end{array}}{\sigma \xrightarrow{\delta} \sigma \oplus \delta} \qquad \frac{\sigma_1 \xrightarrow{\delta} \sigma_1' \quad \sigma_2 \xrightarrow{\delta} \sigma_2'}{\sigma_1 \cup \sigma_2 \xrightarrow{\delta} \sigma_1' \cup \sigma_2'}$$

Finally, consider only transitions outgoing from *complete* configurations.

4 Final remarks

For sake of clarity we restrict the semantic model to the minimal set of primitives for allowing for inter-process communication, process instance creation and destruction. Nevertheless, with slight extensions we can cover all the features of the common language, in particular:

- *saved signals*: mask the queue content and consider always the first visible signal only;

- explicit time-progress conditions (time invariants) in states: they can be taken into account easily when compute time progress;
- *external code integration*: as long as they operates on usual data (excluding clocks) procedures are just black-box configuration transformers;
- *external signalroutes*: an *open* system semantics could be given by considering that input-labelled transitions are triggered by signals coming from the environment;
- *shared variables*: an explicit read/write (or import/export) mechanism can be described