Scaling-Up Behavioral Programming: Steps from Basic Principles to Application Architectures

Supplementary Material

David Harel
Weizmann Institute of Science
Rehovot, Israel
david.harel@weizmann.ac.il

Guy Katz
Weizmann Institute of Science
Rehovot, Israel
guy.katz@weizmann.ac.il

I. SEMANTICS

Having informally described the extensions we propose for the traditional BP framework, this section is dedicated to their rigorous formulation. The definitions are based on, and are similar to, those described in Section 2 of the paper, with alterations to accommodate parameterized events, customizable event selection and dynamic thread creation. The more extensive changes are meant to support the notion of time; in particular, we no longer assume that threads transitions occur instantly, and introduce special "synchronization transitions" and "timeout transitions".

I.1 Event Sets

As discussed in Section 6 of the paper, we want our new semantics to support unbounded parameterized events. For simplicity, we assume that the domains of these parameters are enumerable (say, integers or strings), and so the set of all possible parameterized events E is also enumerable. We assume that this set does not contain the two special symbols: $\perp$, denoting a timeout, and $\top$, denoting thread synchronization.

I.2 Behavior Threads

Let E be an event set and let $BT = \{BT^1, BT^2, \ldots\}$ denote a (possibly infinite) set of threads. These threads can be thought of as *templates*, instances of which are spawned as the program runs. In particular, multiple instances of the same thread may exist simultaneously. Each thread is formalized by the tuple $BT^i = \langle Q^i, q_0^i, \delta^i, \xi^i, R^i, B^i, T^i \rangle$, where Q^i is the (possibly infinite) set of states, q_0^i is an initial state, and $R, B : Q^i \to 2^E$ are functions that map states to requested and blocked states (respectively), as before. As discussed in Section 6 of the paper, the range of B^i may contain infinite sets, but the range of R^i may only contain finite sets.

$T^i : Q^i \to (0, \infty)$ is a timeout function, assigning each state a positive timeout value. This value is not an absolute time, but rather the amount of (say, seconds) the thread is willing to spend in that state before a timeout should occur.

The transition relation $\delta^i \subseteq Q^i \times (E \cup \{\perp\}) \times Q^i$ is used to map states and events to new states. We stipulate that for every state q for which $T^i(q) < \infty$ there is an edge $\langle q, \perp, \tilde{q} \rangle \in \delta^i$. In other words, if a thread declares a timeout, a timeout would cause it to transition.

Finally, $\xi^i : \delta^i \to BT \times \mathbb{N}$ is a function that maps transitions to thread instances that should be spawned when they are traversed. Each such thread is paired with the number of instances that should be spawned. We sometimes abuse notation, and consider ξ^i as mapping states to multisets.

Observe that the above definition does not support thread *termination*; indeed, for simplicity, we assume that when a thread terminates it goes into a special "shutdown" state, in which it does not request, wait-for or block any events, and sets its timeout value to ∞. This is the semantic equivalent of thread termination, as the thread can no longer affect the execution; in practice, the thread can safely be discarded.

I.3 Configurations

A *thread configuration* c is given by the tuple

$$c = \langle index, sync, state, time \rangle,$$

where:

- *index* is an integer, denoting that the thread in question is an instance of thread template BT^{index}.
- *sync* is a boolean variable, indicating whether the thread is currently synchronized or not.
- $state \in Q^{index}$ indicates the state the thread is in if $sync = true$, or the state the thread is expected to reach in its next synchronization if $sync = false$.
- *time* is a positive real, indicating the instant in time when the thread last synchronized. This field is only meaningful if $sync = true$.

Two thread configurations are equal if and only if all of their meaningful fields are equal. A *system configuration* γ is a finite tuple $\gamma = \langle c^1, \ldots, c^k, t \rangle$, where $k \geq 1$ and each $c^i = \langle index^i, sync^i, state^i, time^i \rangle$ is a thread configuration, and t is a positive real, indicating the current time. The system configuration indicates which thread instances are currently active, and thus indicates the global configuration of the system.

A system configuration γ is called *initial* if and only if:

$$\forall_{1 \leq i \leq k}, \left(sync^i = false \ \wedge \ state^i = q_0^{index^i} \right) \bigwedge t = 0$$

That is, all currently running thread instances are unsynchronized, are expected to arrive at their initial states in their next synchronization, and the system time is 0. This is the state of the system at the beginning of the execution.

Given two configurations $\gamma = \langle c^1, c^2, \ldots, c^k, t\rangle$ and $\tilde{\gamma} = \langle \tilde{c}^1, \tilde{c}^2, \ldots, \tilde{c}^{k'}, \tilde{t}\rangle$ and an event $e \in E \cup \{\bot, \top\}$, we say that $\tilde{\gamma}$ is a successor of γ with respect to e, denoted $\gamma \xrightarrow{e} \tilde{\gamma}$, if the following conditions apply:

1. Existing threads are preserved: $k' \geq k$ and
$$\forall_{1\leq i\leq k},\ index^i = \widetilde{index}^i.$$
2. Time moves forward: $\tilde{t} \geq t$. We assume event selection can be resolved in zero time, and allow $\tilde{t} = t$.
3. If $e = \top$, then $\gamma \xrightarrow{e} \tilde{\gamma}$ is a valid *synchronization transition*; if $e = \bot$, then it is a *timeout transition*; and otherwise, it is an *event selection transition*. These terms are defined next.

I.3.1 Synchronization Transitions

These are transitions for which $e = \top$; they correspond to a single thread synchronizing. A transition is a valid synchronization transition if the following conditions hold:

- Synchronization transitions cannot spawn new threads, i.e. $k = k'$.
- At least one thread is unsynchronized, i.e. $\exists_{1\leq i\leq k}$ such that $sync^i = false$.
- No synchronized threads have timed-out:
$$\forall_{1\leq j\leq k},\ \left(sync^j = true \implies T^j(state^j) + time^j < \tilde{t}\right)$$
- All threads except the thread that timed-out remain in the same configuration: $\forall_{1\leq j\leq k},\ \left(j \neq i \implies c^j = \tilde{c}^j\right)$.
- The thread that synchronized is updated to indicate that it has arrived at its expected state in that instant: $\widetilde{sync}^i = true \ \wedge\ \widetilde{state}^i = state^i \ \wedge\ \widetilde{time}^i = \tilde{t}$.

I.3.2 Event Selection Transitions

These are transitions with "real" events, i.e. $e \in E$. They are allowed only when the following conditions hold:

- All threads are synchronized, i.e. $\forall_{1\leq i\leq k},\ sync^i = true$.
- The transition must occur immediately upon the last synchronization event: $\tilde{t} = \max_{1\leq i\leq k}\{time^i\}$.
- Event e must be enabled: $e \in \bigcup_{i=1}^{n} R^{index^i}(state^i) - \bigcup_{i=1}^{n} B^{index^i}(state^i)$.
- As a result of the event being triggered, all threads that have transitions for the event become unsynchronized, and their new expected states (when next they synchronize) are determined according to their transition rules:
$$\forall_{1\leq i\leq k},\ (\delta^i(state^i, e) \neq \emptyset \implies \widetilde{sync}^i = false \ \wedge\ \widetilde{state}^i \in \delta^i(state^i, e))$$
- Threads that did not have a transition to traverse retain their configurations:
$$\forall_{1\leq i\leq k},\ \left(\delta^i(state^i, e) = \emptyset \implies c^i = \tilde{c}^i\right)$$
- New threads may be spawned as a result of the transition. These threads correspond to thread configurations $\tilde{c}^{k+1}, \ldots, \tilde{c}^{k'}$. These threads must be precisely those threads that the existing threads spawn, i.e.
$$\{\widetilde{index}^{k+1}, \ldots, \widetilde{index}^{k'}\} = \bigcup_{1\leq i\leq k \ \mid\ \delta^i(state^i,e)\neq\emptyset} \xi^{index_i}(\langle state^i, e, \widetilde{state}^i\rangle)$$
where both hands of the equation are interpreted as multisets.
- New threads are spawned unsynchronized, and are expected to reach their initial states:
$$\forall_{k<i\leq k'},\ \left(\widetilde{sync}^i = false \ \wedge\ \widetilde{state}^i = \widetilde{q_0^{index^i}}\right).$$

I.3.3 Timeout Transitions

These are transitions with $e = \bot$. They occur when a previously synchronized thread times out. Formally, they are allowed if and only if the following holds:

- Timeout transitions are only allowed when no *event selection transitions* are enabled; that is, if there exists at least one unsynchronized thread or if all threads are synchronized but there are no enabled events. Formally, denoting by E the set of enabled events
$$E = \bigcup_{i=1}^{k} R^{index^i}(state^i) - \bigcup_{i=1}^{k} B^{index^i}(state^i),$$
we stipulate that $\left(\exists_{1\leq i\leq k},\ sync^i = false\right) \vee (E = \emptyset)$.
- One thread has to have timed out, i.e
$$\exists_{1\leq i\leq k}, \left(sync^i = true \ \wedge\ T^i(state^i) + time^i = \tilde{t}\right)$$
- This thread becomes unsynchronized and traverses a timeout transition:
$$\widetilde{sync}^i = false \ \wedge\ \widetilde{state}^i \in \delta^i(state^i, \bot)$$
- All other threads remain in the same configurations:
$$\forall_{1\leq j\leq k}\left(j \neq i \implies c^j = \tilde{c}^j\right)$$
- The transition may spawn new threads:
$$\{\widetilde{index}^{k+1}, \ldots, \widetilde{index}^{k'}\} = \xi^{index^i}(\langle state^i, \bot, \widetilde{state}^i\rangle)$$
- New threads are spawned unsynchronized, and are expected to reach their initial states:
$$\forall_{k<j\leq k'},\ \left(\widetilde{sync}^j = false \ \wedge\ \widetilde{state}^j = \widetilde{q_0^{index^j}}\right).$$

I.3.4 Thread Termination

For simplicity, the concept of thread termination is not included in the semantics. Instead, we assume that terminated threads enter a dormant state, in which they request, wait-for and block nothing, and do not specify a timeout. This is the equivalent of removing the thread from the pool of active threads. Naturally, in practice it is better to let threads terminate and free the resources they were allocated. Indeed, this is the case in BPC.

I.3.5 System Vs. Environment

From the ESM's point of view, synchronization transitions can be seen as managed by the environment; the ESM has no control on when threads will synchronize. The other two kinds — event selection transitions and timeout transitions — are triggered by the ESM, and it has no flexibility in selecting and scheduling them. As previously mentioned, it would be interesting to extend this work to allow the ESM flexibility in, say, purposely delaying event selection transitions, in order to achieve some goal, along the outline of the smart play-out mechanism of [2, 3].

I.4 Behavioral Programs

A behavioral program P consists of a (possibly infinite) event set E, a (possible infinite) thread template set $BT = \{BT^1, BT^2, \ldots\}$ an initial system configuration γ^0, and a event selection strategy f_{es} (as defined in Section 4 of the paper).

An execution ρ of P is a sequence $\rho = \gamma^0 \xrightarrow{e_0} \gamma^1 \xrightarrow{e_1} \ldots$ of successive configurations, where $e_i \in E \cup \{\bot, \top\}$ for all i. For every $e_i \notin \{\top, \bot\}$, we require that $f_{es}(\gamma_0, \gamma_1, \ldots, \gamma_i) = e_i$, that is that the triggering of "real" events is performed according to the strategy. The execution may either be infinite, or finite if it ends in a *terminal* configuration — a configuration with no successors. Specifically, a terminal configuration is one in which all threads have synchronized, there are no enabled events, and all threads have set their timeout values to ∞.

The *run* that corresponds to execution ρ is a (possibly infinite) sequence of event-and-time pairs $\langle\langle e_{i_0}, t_{i_0}\rangle, \langle e_{i_1}, t_{i_1}\rangle, \ldots\rangle$ that correspond to just the *event selection transitions* of ρ. The run indicates the "real" events that were triggered, and the time of their triggering. Timeout and synchronization transitions are considered internal, and do not appear in the run. The language of a behavioral program P, denoted $\mathfrak{L}(P)$, is the set of runs of all valid executions of the system.

Note. The above semantics can be extended to support sensor threads that do not delay the system as they wait for input, along the lines of [1]. Intuitively, this is performed by relaxing the prerequisites of *event selection transitions*, to no longer require that the sensor threads be synchronized; the details are omitted. Extending the formalism to fully support the eager execution mechanism of [1] in the presence of timeouts is left for future work.

II. REFERENCES

[1] D. Harel, A. Kantor, and G. Katz. Relaxing Synchronization Constraints in Behavioral Programs. In *Proc. 19th Int. Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, pages 355–372, 2013.

[2] D. Harel, H. Kugler, R. Marelly, and A. Pnueli. Smart Play-Out of Behavioral Requirements. In *Proc. 4th Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 378–398, 2002.

[3] D. Harel, H. Kugler, and A. Pnueli. Smart Play-Out Extended: Time and Forbidden Elements. In *Proc. 4th Int. Conf. on Quality Software (QSIC)*, pages 2–10, 2004.