On Composing and Proving the Correctness of Reactive Behavior

Supplementary Material

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

Amir Kantor
Weizmann Institute of Science
Rehovot, Israel
amir.kantor@weizmann.ac.il

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

Assaf Marron
Weizmann Institute of Science
Rehovot, Israel
assaf.marron@weizmann.ac.il

Lior Mizrahi
Ben-Gurion University
Beer-Sheva, Israel
liormizr@cs.bgu.ac.il

Gera Weiss
Ben-Gurion University
Beer-Sheva, Israel
geraw@cs.bgu.ac.il

Appendix 1. Z3 axioms for BP with priorities

As stated in Section 3, the Z3 formulation of the BP axioms can be altered to accommodate other variants of event-selection mechanisms. Below, we give the axioms for a priority-based scheme: each event is requested with an integer representing its priority, and among all events that are requested and not blocked the one of highest priority is selected for triggering.

The trace function includes the priority of the event triggered at each step:

```
Priority   = IntSort();

requested  = Function('requested', Event,
             Time, Priority, BoolSort())

blocked    = Function('blocked', Event,
             Time, BoolSort())

TraceEntry = Datatype('TraceEntry')
TraceEntry.declare('TEntry',
                   ('event', Event),
                   ('priority', Priority))
TraceEntry = TraceEntry.create()

trace      = Function('trace', Time,
             TraceEntry)
```

The axioms describe priority-based selection:

```
∀e, t, p : ¬requested(e, t, p) ⇒
           trace(t) ≠ TraceEntry(e,p)

∀e, t    : blocked(e,t) ⇒ event(trace(t)) ≠ e

∀e, t, pr: requested(e,t,pr) ∧ ¬blocked(e,t) ⇒
           priority(trace(t)) ≥ pr
```

Finally, the `requested_by` helper function handles priorities:

$$\texttt{requested(e,t,pr)} \Leftrightarrow \bigvee_{\texttt{bt}\in\texttt{BThreads}} \texttt{requested\_by(e,t,pr,bt)}$$

The `blocked_by` helper function remains unchanged.

These axioms can be used to verify properties of priority-based BP programs in much the same way as shown in the examples of Section 4.

Appendix 2. Deadlock-freedom of the application in Sec. 4.4

Below is a manual proof that the specification of the dining philosophers problem with one left-handed philosopher is deadlock-free. The proof is independent of the number of philosophers.

Proof. Let P_n be the left-handed philosopher.

1. If any philosopher b-thread is in an "Eating" state or "Put down one fork" state, the next event associated with this philosopher is not blocked by any b-thread. Consequently, the system is not deadlocked.
2. Hence, if there is a deadlock, all philosopher b-threads are either in a "Thinking" state or in a "Picked up one fork" state.
3. Thus, P_n is not holding F_0.
4. If P_0 is in "Picked up one fork" state, then she can pick up F_0 (the event is requested and not blocked).
5. If P_0 is in "Thinking" state, then (as P_1 is either in "Thinking" or in "Picked up one fork" state) she can pick up F_1 (the event is requested and not blocked).
6. Hence, the system is not deadlocked.

□