Algorithm to Create the PLTS for Memory-Related Deadlocks

September 7, 2021

The pseudo-code to create a PLTS $L = \langle \mathcal{Q}, q_{start}, \mathcal{A}, \mathcal{L} \rangle$ of a mapped workload $\langle \mathcal{T}, \mathcal{B}, \mathcal{D} \rangle$ is given in Algorithm 1.

Algorithm 1 visits all states that have not been visited yet and, for each of them, adds to $\mathcal{L}$ new transitions corresponding to valid task start (lines 11 to 27) or end (lines 28 to 40) actions.

If the arrival state of a new transition is new (lines 13 and 30) its memories occupancy is computed (lines 14 to 18 and 31 to 35).

Of course, in a state q, a task can be started only if was a future task in state q, all its immediate predecessor tasks ended (line 11) and all its output buffers can be allocated without overflowing any memory (lines 19 and 24). Similarly a task t can be ended only if it was running (line 28).

For $\mathbf{start}(t)$ actions the new memories occupancy is computed by considering all output buffers of task t (line 15) and adding their size to the occupancy of their memory (line 17). For $\mathbf{end}(t)$ actions the new memories occupancy is computed by identifying the input buffers of task t that can be deallocated because all their consumer tasks ended (line 32) and subtracting their size from the occupancy of their memory (line 34).

```
1  Function buildPLTS(D = ⟨T, B, D⟩) is
2      q_start ← (∅, ∅, T); O_{q_start} ← 0^n;                    // initial state
3      q_end ← (T, ∅, ∅); O_{q_end} ← 0^n;                        // final state
4      Q ← {q_start, q_end};                                      // initialize set of states
5      A ← {start(t), end(t) | t ∈ T};                            // set of actions
6      L ← ∅;                                                     // initialize set of transitions
7      V ← {q_start};                                             // initialize set of states to visit
8      while V ≠ ∅ do                                             // while there are states to visit
9          select q ∈ V;                                          // pick state q to visit
10         V ← V \ {q};                                           // mark q as visited
           // for future tasks which predecessors all ended
11         foreach t ∈ F_q such that P_t ⊂ E_q do
               // build state r from q by starting task t
12             r ← (E_q, R_q ∪ {t}, F_q \ {t});
13             if r ∉ Q then                                      // if r new
                   // compute memories occupancy
14                 O_r ← O_q;                                     // initialization
15                 foreach b ∈ O_t do                             // for output buffers of t
16                     i ← idx(b);                                // b memory index
17                     O_r[i] ← O_r[i] + size(b);                 // allocate buffer
18                 end
19                 if O_r ≤ X then                                // if no memory overflow
20                     Q ← Q ∪ {r};                               // add new state
21                     V ← V ∪ {r};                               // must visit new state
22                 end
23             end
24             if O_r ≤ X then                                    // if no memory overflow
25                 L ← L ∪ {q --start(t)--> r};                   // add new transition
26             end
27         end
28         foreach t ∈ R_q do                                     // for running tasks
               // build state r from q by ending task t
29             r ← (E_q ∪ {t}, R_q \ {t}, F_q);
30             if r ∉ Q then                                      // if r new
                   // compute memories occupancy
31                 O_r ← O_q;                                     // initialization
                   // for input buffers which consumers all ended
32                 foreach b ∈ I_t such that C_b ⊂ E_r do
33                     i ← idx(b);                                // b memory index
34                     O_r[i] ← O_r[i] − size(b);                 // deallocate buffer
35                 end
36                 Q ← Q ∪ {r};                                   // add new state
37                 V ← V ∪ {r};                                   // must visit new state
38             end
39             L ← L ∪ {q --end(t)--> r};                         // add new transition
40         end
41     end
42     return L = ⟨Q, q_start, A, L⟩;                             // return PLTS
43 end
```

Algorithm 1: Algorithm to construct the PLTS