CSL Technical Report •

Contents

```
sal > (display-curr-states)
State 1
--- Input Variables (assignments) ---
request = true;
--- System Variables (assignments) ---
state = ready;
-----------------------------
State 2
--- Input Variables (assignments) ---
request = false;
--- System Variables (assignments) ---
state = ready;
-----------------------------

sal > (select-state! 1)

sal > (display-curr-states)
State 1
--- Input Variables (assignments) ---
request = true;
--- System Variables (assignments) ---
state = ready;
-----------------------------
```

Command (`step!`) performs a simulation step, that is, it appends the successors of the set of current states in the current trace. Clearly, the set of current states is also updated.

```
sal > (step!)

sal > (display-curr-trace)
Step 0:
--- Input Variables (assignments) ---
request = true;
--- System Variables (assignments) ---
state = ready;
------------------------
Step 1:
--- Input Variables (assignments) ---
request = false;
--- System Variables (assignments) ---
state = busy;
```

Command (`filter-curr-states! <constraint>`) provides an alternative way to selt-41equestto cseto yd y cult-4rento Ttra267(th)051lt-49setod

Chapter 3

The Peterson Protocol

Chapter 4

The Bakery Protocol

In this chapter, we specify the bakery protocol. The SAL files for this example are located in the following subdirectory in the SAL distribution package: `examples/bakery`

```
min_non_zero_ticket_aux(rsrc : RSRC, idx : Job_Idx) : Ticket_Idx =        6
  IF idx = N THEN rsrc.data[idx]
  ELSE LET curr: Ticket_Idx = rsrc.data[idx],F          rest: Ticket_Idx = min_non_zero_ticket_aux(rsrc, idx +
  ELSE minrc, rest) ENDIFENDIF;min_non_zero_ticket(rsrc : RSRC) : Ticket_Idx =
    min_non_zero_ticket_aux(rsrc, 1);can_enter_critical?(rsrc : RSRC, job_idx : Job_Idx): BOOLEAN =
      LET min_ticket: Ticket_Idx = min_non_zero_ticket(rsrc),F          job_ticket: Ticket_Idx = rsrc.data[job_i
        rsrc.next_ticket = B + 1;next_ticket(rsrc : RSRC, job_idx : Job_Idx): RSRC =
          IF saturated?(rsrc) THEN rsrc
          ELSE (rsrc WITH .data[job_idx] := rsrc.next_ticket)          WITH .next_ticket := rsrc.next_ticket +
            rsrc WITH .data[job_idx] := 0;can_reset_ticket_counter?(rsrc : RSRC): BOOLEAN =
```

value of `pc` is `sleeping`

`Job_Idx` is a subrange `[1..N]`. Notice that each instance of `job` is initialized

for safety properties is

This property states that every job trying to enter the critical section will eventually succeed. The following command can be used to prove the property:

---

circular shift register which shifts up one place each clock cycle. Each cell also has a local variable `w` (*waiting*

```
aux_module : MODULE =                                         13
BEGIN
  OUTPUT zero_const : BOOLEAN
  INPUT aux : BOOLEAN
  OUTPUT inv_aux : BOOLEAN
  DEFINITION
    zero_const = FALSE;
    inv_aux = NOT(aux)
END;

arbiter: MODULE =
  WITH OUTPUT Ack : Array;
       INPUT Req : Array;
       OUTPUT Token : Array;
       OUTPUT Grant : Array;
       OUTPUT Override : Array
    (RENAME aux TO Override[n], inv_aux TO Grant[n]
     IN aux_module)
```

```
at_most_one_ack:
  THEOREM arbiter |- G((FORALL (i : [1..n - 1]):
```

Inspecting the counterexample, you can notice that more than one cell has the token. So, we may use the following auxiliary lemma to prove the property `at_most_one_ack`.

```
at_most_one_token:
  THEOREM arbiter |- G((FORALL (i : [1..n - 1]):
                        (FORALL (j : [i + 1..n]):
```