Nicolas Desfeux Student No :110477367

A model Checking Exercise : Pablo's restaurant

Newcastle University

December 2, 2011

a) Simple promela design

i) Protocol implementation

Implementing this protocol requires some global variables. We used a mtype to define the different meal available :

```
mtype = {starter ,main ,desert ,drink}
```

We decided to use two channels :

```
chan order_make = [20] of { mtype, int };
chan service_channel = [20] of { mtype, int };
```

Both channels take value of type {mtype, int}, in order to couple a customer and his order. By this add, we have a way to link a meal with a customer. The first channel is use to send order from the customer to the chief. The second is use to send meal from the chief to the client.

Chief proctype The chief proctype is actually quiet simple. We created a loop that check the order_make channel. As soon as something appears in this channel, the chief can start cooking. And as he cooks very fast, the meal is added to the service_channel channel directly. We associate to this meal the id we received from the order_make channel. The loop structure is done as the chief can only cook one meal at the time.

Customer proctype The customer proctype is a bit more complicate. The customer can go throw different states. The proctype we design consist on a series of switch which leads to different actions. As for the chief, the customer have a loop, as he can order several meal, or leave. The first switch we used check if the client already make an order.

- if the client already make an order:
- if the client haven't order yet :

Runing the model To run the model, we created an init proctype, who launch one chief proctyp, and as many customer proctyp as we need. You can change the number of customer created by changing the variable nb_customer. There is no stop condition.

ii) Check for deadlock, and model checking

We added a end label at the loop in the chief proctyp. That allow the chief not ending at the end of the execution. By this way, we avoid something seen as a mistake by Spin, but which is not in the real life (The chief only stop waiting at the close of the restaurant, that's include a time handling that we didn't implement here). At the end of the execution, each customers have to reach their end (we can't close the restaurant if some customers are still inside !).
To check the model, we run some simulations using the Simulate / Replay function of ispin (with random and interactive progress).
Runing verification confirm that the model we design have no dead lock. We run the safety verification, focusing on invalide endstates (deadlocks). You can the result of this verification in the appendix (figure 1 page v).

b) Adding properties

i) Maximum order and choices constraint

ii) Assertion

c) Additional possibility

i) Thinking state

ii) Checking for deadlock

Figure 1: Verification on simple model implementation

```
verification result:
spin -a  Pablo-a.pml
gcc-4 -DMEMLIM=1024 -O2 -DXUSAFE -DSAFETY -DNOCLAIM -w -o pan
    pan.c
./pan -m10000  -A
Pid: 4596
error: max search depth too small

(Spin Version 6.1.0 -- 4 May 2011)
  + Partial Order Reduction

Full statespace search for:
  never claim           - (not selected)
  assertion violations  - (disabled by -A flag)
  cycle checks          - (disabled by -DSAFETY)
  invalid end states    +

State-vector 368 byte, depth reached 9999, errors: 0
    55727 states, stored
    44294 states, matched
   100021 transitions (= stored+matched)
        0 atomic steps
hash conflicts:      1431 (resolved)

   22.754 memory usage (Mbyte)

unreached in proctype chief
  Pablo-a.pml:25, state 9, "-end-"
  (1 of 9 states)
unreached in proctype customer
  (0 of 38 states)
unreached in init
  (0 of 10 states)

pan: elapsed time 0.096 seconds
No errors found -- did you verify all claims?
```