The dining philosophers, a case study for Quasar

Evangelista Sami, Kaiser Claude Pradat-Peyre Jean-François, Rousseau Pierre

22 mai 2003

This case study for the verification tool Quasar presents a solution to the well known problem of the dining philosophers initially introduced by Dijkstra.
This one has been given by Laurent Pautet in an advanced computing science course at ENST Paris. Laurent Pautet solves the problem by a subtle use of the requeue mechanism. This results in a non trivial solution that human reasoning can't easily handle. However, the use of a verification program such Quasar, which is based on the Petri net formalism, enables us to formally prove some critical properties like the deadlock presence.
On this example, Quasar reports no deadlock after a few seconds search. This is a due to the egg-shell model semantic of the protected objects.
A small modification of the program, for example changing the following assignement :

Updating := Get_Left_Q'count > 0 **or** Get_Right_Q'count > 0

at line 41 by this one :

Updating := Get_Left'count > 0 **and** Get_Right'count > 0

leads to a "deadlockable" program that even many simulations didn't report because of the queuing policy implemented. Using Quasar on the modified program show us the execution which leads on the deadlock. By this way, we can prove that this solution doesn't allow any modification.

```
1  procedure Dining_Philosophers is
2     type Id is mod 5;
3     type Ready_Table is array (Id) of Boolean;
4
5     -- STICKS --
6     protected Sticks is
7        entry Get_Left (C : in Id);
8        entry Get_Right (C : in Id);
9        procedure Free (C : in Id);
10       entry Get_Left_Q (C : in Id);
11       entry Get_Right_Q (C : in Id);
12    private
13       Ready : Ready_Table := (others => true);
14       Updating : Boolean := False;
15    end Sticks;
16    protected body Sticks is
17       entry Get_Left(C : in Id) when not Updating is
18       begin
19          if Ready(C) then
20             Ready(C) := False;
21             requeue Get_Right;
22          else
23             requeue Get_Left_Q;
24          end if;
25       end Get_Left;
26       entry Get_Left_Q(C : in Id) when Updating is
27       begin
28          Updating := Get_Left_Q'count > 0 or Get_Right_Q'count > 0;
29          requeue Get_Left;
30       end Get_Left_Q;
31       entry Get_Right(C : in Id) when not Updating is
32       begin
33          if Ready(C+1) then
34             Ready(C+1) := False;
35          else
36             requeue Get_Right_Q;
37          end if;
38       end Get_Right;
```

```
39          entry Get_Right_Q(C : in Id) when Updating is
40          begin
41             Updating := Get_Left_Q'count > 0 or Get_Right_Q'count > 0;
42             requeue Get_Right;
43          end Get_Right_Q;
44          procedure Free(C : in Id) is
45          begin
46             Ready(C) := True;
47             Ready(C+1) := True;
48             Updating := Get_Left_Q'count > 0 or Get_Right_Q'count > 0;
49          end Free;
50       end Sticks;
51       --=========--
52
53       -- PHILOSOPHERS --
54       task type Philosopher is
55          entry Init (N : in Id);
56       end Philosopher;
57       task body Philosopher is
58          Self : Id;
59       begin
60          accept Init(N : in Id) do
61             Self := N;
62          end Init;
63          loop
64             Sticks.Get_Left(Self);
65             Sticks.Free(Self);
66          end loop;
67       end Philosopher;
68       type Tab is array (Id) of Philosopher;
69       Philosophers : Tab;
70       --================--
71
72    begin
73       for I in Id loop
74          Philosophers(I).Init(I);
75       end loop;
76    end Dining_Philosophers;
```