Formal Models for Parallel and Distributed Systems Exercise 2 (June 19)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

June 9, 2017

The exercise is to be submitted by the deadline stated above via the Moodle interface as a single .zip or .tgz file containing

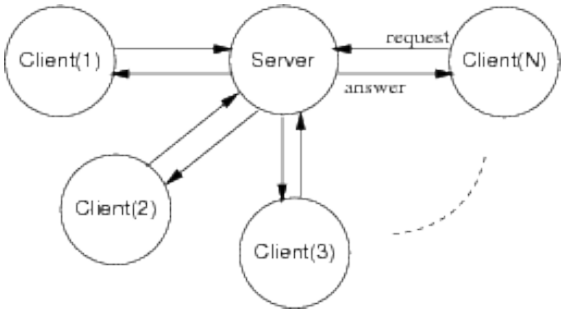
1. a PDF file with a decent cover page (mentioning the title of the course, your full name and Matrikelnummer) with
   - listings of the model files and
   - the outputs/screenshots of the tool,
2. the model files used in the exercise.

A Client/Server System

Take a distributed system of a server and N clients numbered $1, \ldots, N$ where the server maintains a shared resource which it grants to at most one of the clients at a time, as depicted by the following pseudo-code:

```
Server:
  local given, waiting, sender
begin
  given := 0; waiting := { }
  loop
    sender := receiveRequest()
    if sender = given then
      if waiting = { } then
        given := 0
      else
        choose given from waiting
        waiting := waiting \ { given }
        sendAnswer(given)
      endif
    elsif given = 0 then
      given := sender
      sendAnswer(given)
    else
      waiting :=
        waiting U { sender }
    endif
  endloop
end Server
```

```
Client(p):
  param ident
begin
  loop
    ...
    c1: sendRequest()
    c2: receiveAnswer()
    ... // critical region
    c3: sendRequest()
  endloop
end Client
```



Develop a CCS specification (in the value passing calculus presented in the lecture) of the system with one server and N clients where the server and the clients interact by synchronous message passing. You may use in your specification variables ranging over integers and can use the usual integer operations in conditional expressions and output actions. Please note that a set S of at most N integers $1, \ldots, N$ can be represented by a single N-bit integer whose bit i is set if and only if $i + 1 \in S$. Please note that the server continuously receives a message and then choses one of four possible execution paths (depending on the *sender* of the message and the local state variables *given* and *waiting*); every client has a single execution path of sending, receiving, and again sending a message.

Next translate this specification as directly as possible to a FSP model with $N = 2$ (possibly $N = 3$, if the state space does not get too large).

Construct drawings for the labeled transition system of the server process, one client process, and (if possible) of the composed system.

Construct manually in the animator a trace of a (part of a) system run where Client 1 requests the resource, receives the resource, and releases the resource.

Check whether the system may run into a deadlock and give the output of the check.

Check whether the system maintains liveness for client 1 by defining a progress property that includes the client's action for entering the critical region, e.g.

```
progress LIVENESS = { c[1].enter }
```

(see also example `Twocoin` in LTSA).

Hide from the model all action names except those for entering and exiting the critical region by the clients, perform minimization, and construct a drawing for the minimized system (see also example `User` in LTSA).

Explain whether/how the drawing illustrates that mutual exclusion is preserved.

Check whether the system maintains mutual exclusion by defining a corresponding mutual exclusion property, e.g.

```
property MUTEX = (c[i:1..N].enter->c[i].exit->MUTEX).
```

which is composed with the system (see also Example `Mutex_property` in LTSA).

Also check whether the system maintains mutual exclusion by defining a corresponding FLTL property, e.g.

```
fluent CRITICAL[i:1..N] = < { c[i].enter }, { c[i].exit } >
assert MUTEX = forall[i1:1..N] forall[i2:1..N]
  rigid(i1 < i2) -> [] !(CRITICAL[i1] && CRITICAL[i2])
```

(see also Example `Mutex_fluent` in LTSA).