Formal Models for Parallel and Distributed Systems Exercise 3 (July 17, 2023)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

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

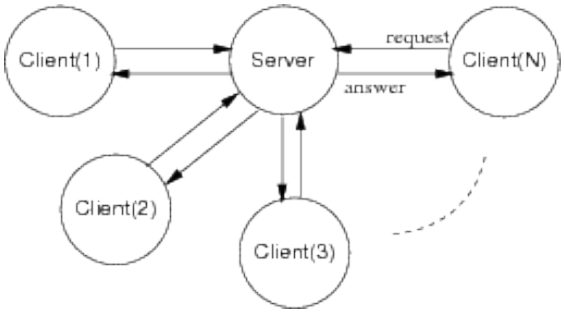
1. a PDF file with a decent cover page (mentioning the title of the course, your full name and Matrikelnummer) with
   - listings of the model files and
   - the outputs/screenshots of the tool,
2. the model files used in the exercise.

LTSA/FSP Model of 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:

```
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 LTSA/FSP model of this system where the server and the clients interact by synchronous message passing (use $N = 2$, possibly $N = 3$, if the state space does not get too large). 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 also 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.

Construct in LTSA 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_P = (enter[i:1..N]-> exit[i] -> MUTEX_P).
```

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

(see also Example `Mutex_fluent` in LTSA).