CMU 17-651 Models of Software Systems Fall 2018

Homework #10: Concurrency

Dario A Lencina-Talarico Due: 05 November 2018

1. Traces and Specifications:

   (a) Enumerate the traces for the following process `P`:

   ```
   P = (a -> a -> END | b -> a -> END).
   ```

   $traces == \{\langle a, a\rangle, \langle b, a\rangle, \langle a\rangle, \langle b\rangle, \emptyset\}$

   (b) Enumerate the traces for the following process `P`:

   ```
   P1 = (a -> a -> END).
   P2 = (a -> b -> END | a -> c -> END).
   ||P = (P1 || P2).
   ```

   $traces == \{\langle a, b\rangle, \langle a, c\rangle, \langle a\rangle, \emptyset\}$

   It is important to mention that P1 seems to always deadlock as it requires 2 a actions to reach the *END* but P2 can only go through 1 a per run.

2. More Concurrency:

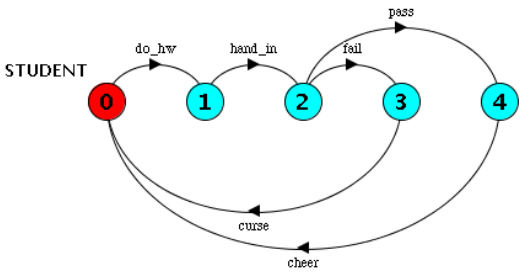
   Consider the two processes `STUDENT` and `TEACHER`, where

   $$\alpha\ \texttt{STUDENT} = \{\texttt{do\_hw}, \texttt{hand\_in}, \texttt{pass}, \texttt{fail}, \texttt{cheer}, \texttt{curse}\}$$
   $$\alpha\ \texttt{TEACHER} = \{\texttt{hand\_in}, \texttt{grade}, \texttt{pass}, \texttt{fail}, \texttt{grumble}\}$$

   The student repeatedly does her homework, hands it in, and gets a pass or fail—cheering when she passes and cursing when she fails. The teacher repeatedly collects the homework, grades it, and then assigns a pass or fail grade—grumbling after any time that he has to give out a failing grade.

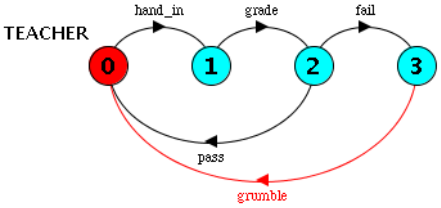
   (a) Write an FSP process that characterizes the student and show a diagram that indicates its behavior.

   ```
   STUDENT = (do_hw -> hand_in -> STUDENT_SEND),

   STUDENT_SEND = (pass -> cheer -> STUDENT | fail -> curse -> STUDENT).
   ```

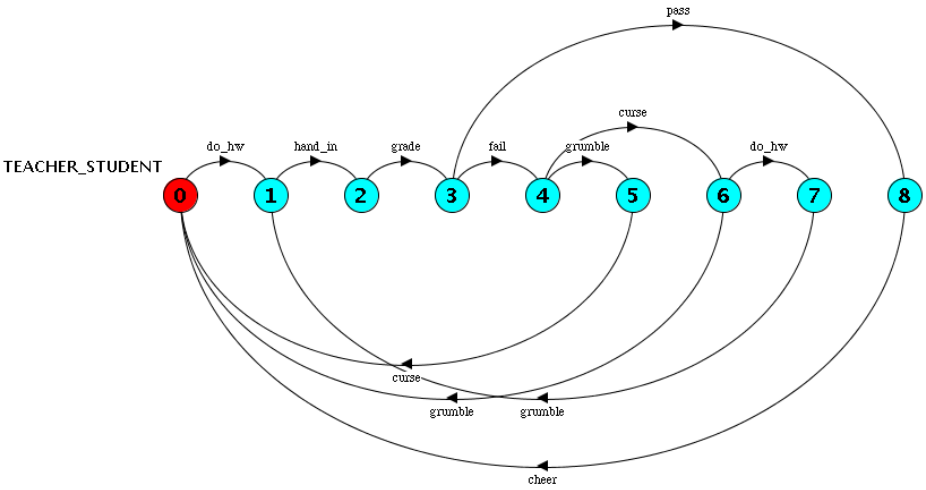


(b) Write an FSP process that characterizes the teacher and show a diagram that indicates its behavior.

```
TEACHER = (hand_in->grade->TEACHER_GRADE_RESULT),

TEACHER_GRADE_RESULT = (pass -> TEACHER | fail -> grumble -> TEACHER).
```

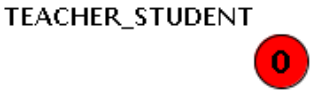


(c) Produce an LTS graph for `STUDENT || TEACHER`.

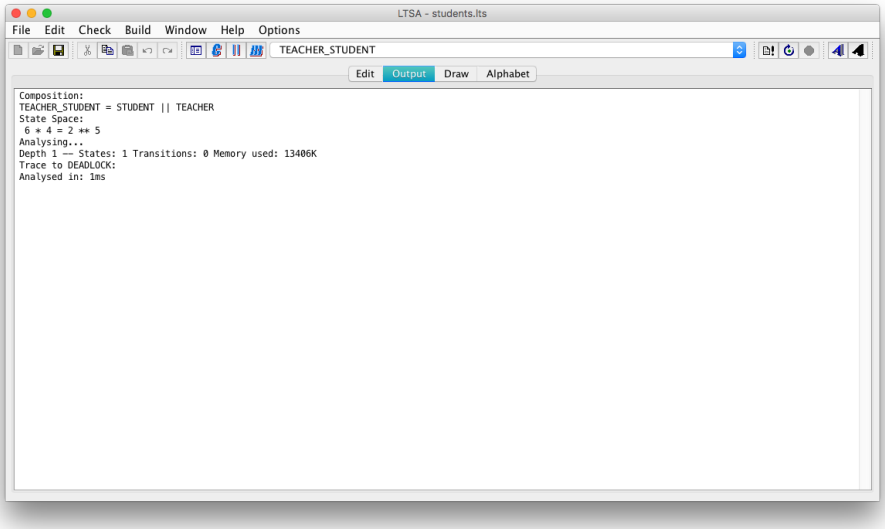


(d) What happens to this process if we augment `STUDENT`'s alphabet with the `grumble` event and have her grumble before doing her homework? Why does this occur?

TEACHER_STUDENT will deadlock because STUDENT will get stuck in grumble waiting for TEACHER to grumble, but TEACHER will get stuck waiting for the student to hand in the homework.



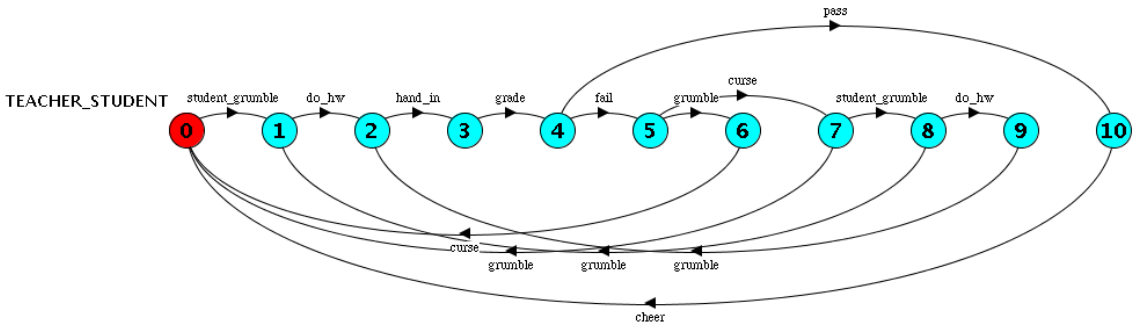
LTSA presents a nice warning message when executing the composition:



(e) If your answer to the previous question involves deadlock, list two ways that you might change the definition to avoid this unintended problem. (NOTE: You may not change the order in which events happen. For example, do not move the student's `grumble` event after her `hand_in` event. Preserve the intended behavior of the model.)

Prefix grumbles with the name of the actor, teacher or student prevents the deadlock.

```
STUDENT = (student.grumble -> do_hw -> hand_in -> STUDENT_SEND),

STUDENT_SEND = (pass -> cheer -> STUDENT | fail -> curse -> STUDENT).

TEACHER = (hand_in->grade->TEACHER_GRADE_RESULT),

TEACHER_GRADE_RESULT = (pass -> TEACHER | fail -> teacher.grumble -> TEACHER).

||RESULT = (STUDENT || TEACHER).
```



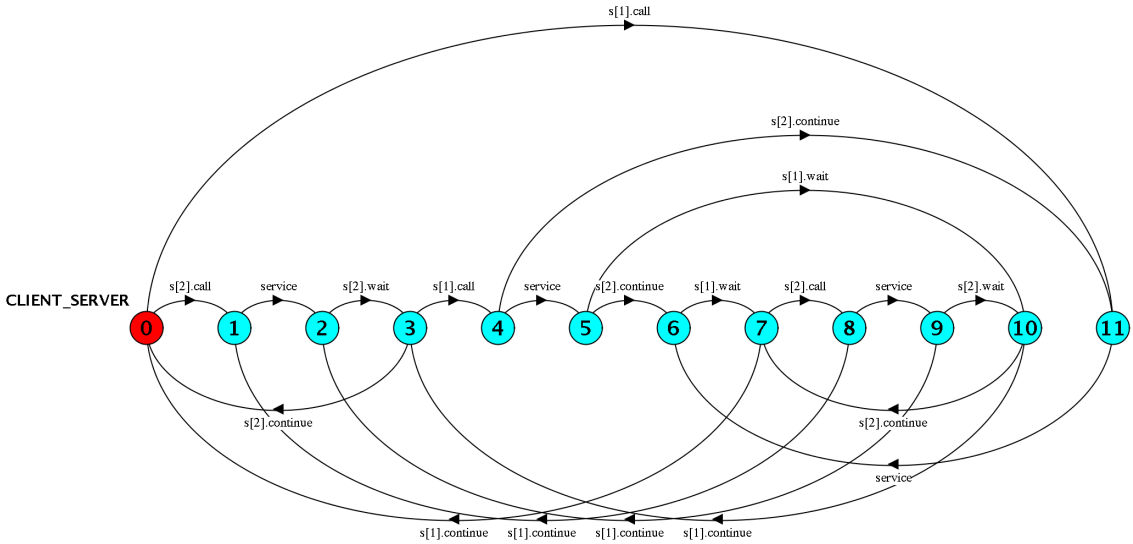
3. Exercises Based on MK06

   Consider the model of the client–server system described in section 3.1.4 of MK06.

   (a) Extend the model of the client–server system so that more than one client can use the server. Your model should support an arbitrary number of clients (`N`).

```
const N = 2

CLIENT = (call->wait->continue->CLIENT).

SERVER = (request->service->reply->SERVER).

||CLIENTS(M = 1) = (forall[i:1..M] s[i]:CLIENT).

||CLIENT_SERVER = (CLIENTS(N) || SERVER)/{{s[1..N]}.call/request, {s[1..N]}.wait/reply}.
```

   Graph for CLIENTS N = 2



(b) Modify your new model of the client–server system so that a client's call may terminate with a `timeout` action rather than a response from the server. (Do not modify the server process.) What condition results from this modification?

My assumption is that if there's a timeout, the client will terminate as opposed to sending another request.

The condition that results is a more realistic implementation of the client server model by incorporating a primitive error scenario.

```
const N = 1

CLIENT = (call->(wait->continue->CLIENT
                 | timeout->ERROR)).

SERVER = (request->service->reply->SERVER).

||CLIENTS(M = 1) = (forall[i:1..M] s[i]:CLIENT).

||CLIENT_SERVER = (CLIENTS(N) || SERVER)/{{s[1..N]}.call/request, {s[1..N]}.wait/reply}.
```