

Homework #10: Concurrency

Dario A Lencina-Talarico

Due: 05 November 2018

1. Traces and Specifications:

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

$$P = (a \rightarrow a \rightarrow \text{END} \mid b \rightarrow a \rightarrow \text{END}).$$

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

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

$$P1 = (a \rightarrow a \rightarrow \text{END}).$$

$$P2 = (a \rightarrow b \rightarrow \text{END} \mid a \rightarrow c \rightarrow \text{END}).$$

$$||P = (P1 \mid P2).$$

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

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

$$\alpha \text{ STUDENT} = \{\text{do_hw}, \text{hand_in}, \text{pass}, \text{fail}, \text{cheer}, \text{curse}\}$$

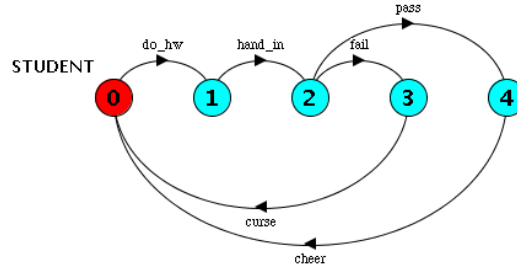
$$\alpha \text{ TEACHER} = \{\text{hand_in}, \text{grade}, \text{pass}, \text{fail}, \text{grumble}\}$$

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.

$$\text{STUDENT} = (\text{do_hw} \rightarrow \text{hand_in} \rightarrow \text{STUDENT_SEND}),$$

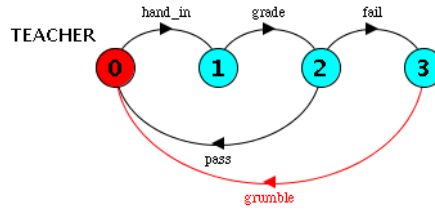
$$\text{STUDENT_SEND} = (\text{pass} \rightarrow \text{cheer} \rightarrow \text{STUDENT} \mid \text{fail} \rightarrow \text{curse} \rightarrow \text{STUDENT}).$$



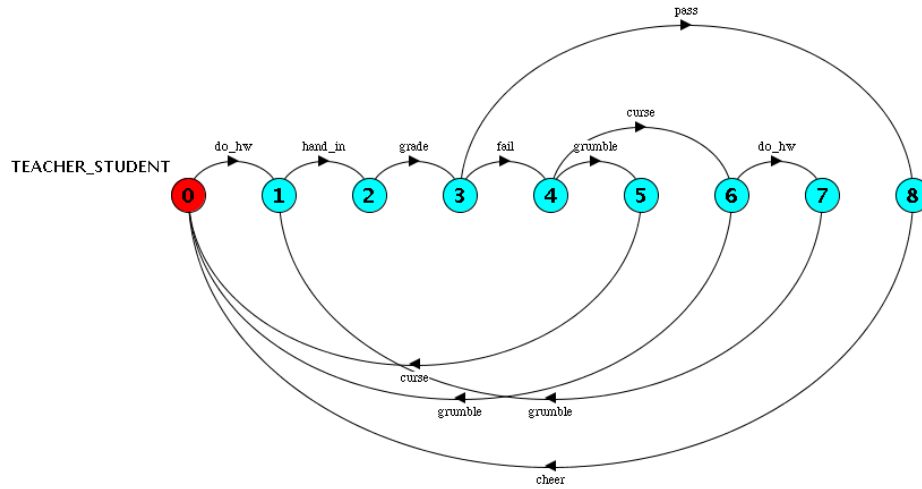
- (b) Write an FSP process that characterizes the teacher and show a diagram that indicates its behavior.

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

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



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



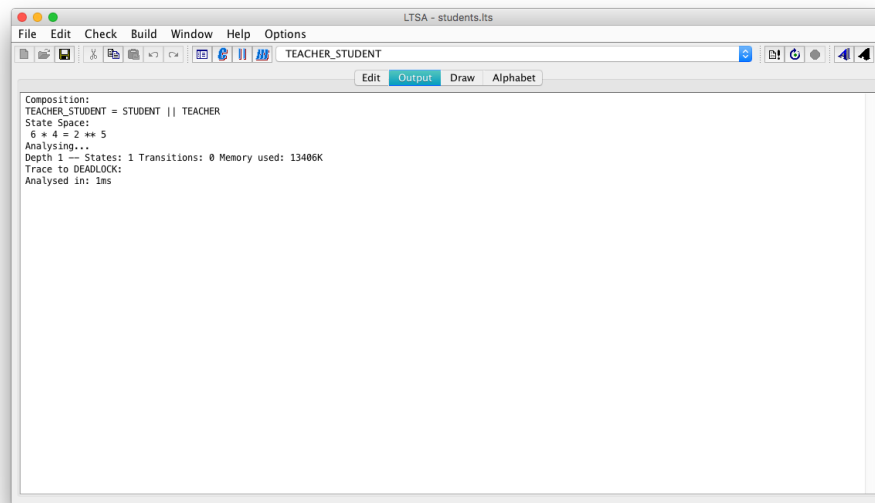
- (d) What happens to this process if we augment `STUDENT`'s alphabet with the `grumble` event and have her grumble before doing her homework? Why does this occur?

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.

TEACHER_STUDENT



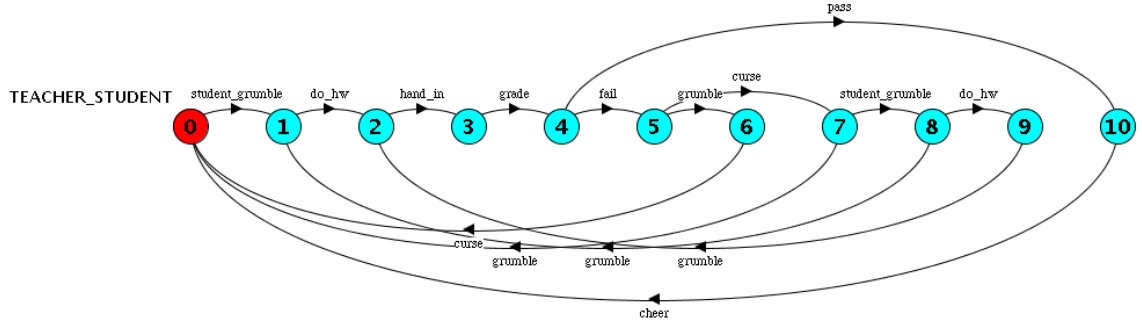
LTSA presents a nice warning message when executing the composition:



(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.)

1. Rename STUDENT grumble to *student_grumble* so that it does not block the TEACHER state machine.
2. Renable TEACHER grumble to *teacher_grumble* so that it does not block the STUDENT state machine.

This is an example of the graph renaming STUDENT grumble to *student_grumble*:



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).

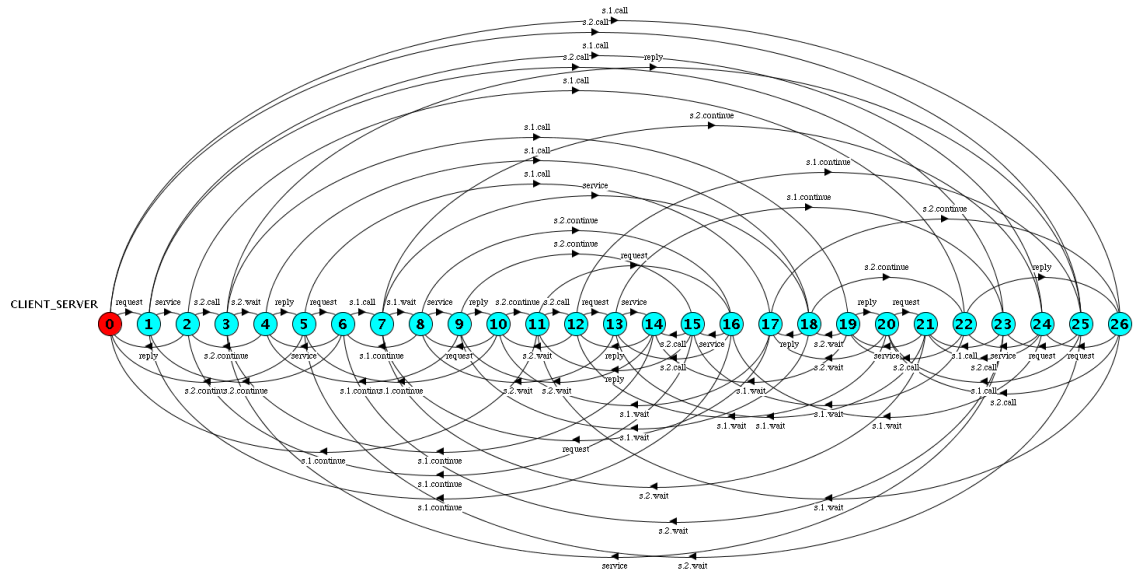
```
CLIENT = (call->wait->continue->CLIENT).
```

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

```
||CLIENTS(N=3) =(forall[i:1..N] s[i]:CLIENT).
```

```
||CLIENT_SERVER = (CLIENTS(2) || SERVER).
```

Graph for CLIENTS N = 2



I could not figure out how to use labeling to synchronize the two state machines, MK06 does not cover how to do so when the processes are created through a set comprehension.

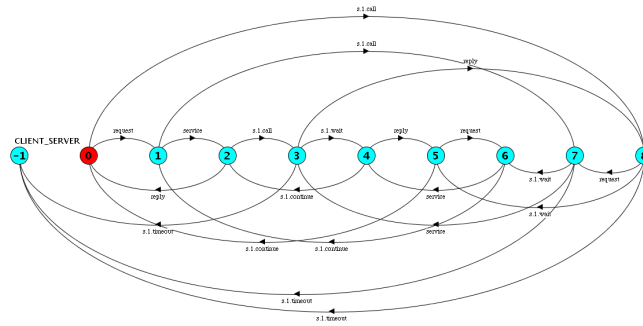
- (b) Modify your new model of the client-server system so that a client's call may terminate with a **timeout** action rather than a response from the server. (Do not modify the server process.) What condition results from this modification?

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 handling.

```
CLIENT = (call->ROBUST_CLIENT),
```

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



Notice that there are some actions that are not possible, for example, that the server gets the request before the client sends it, I could not find a way to silence or remove those actions using the proper labeling to synchronize the two state machines.