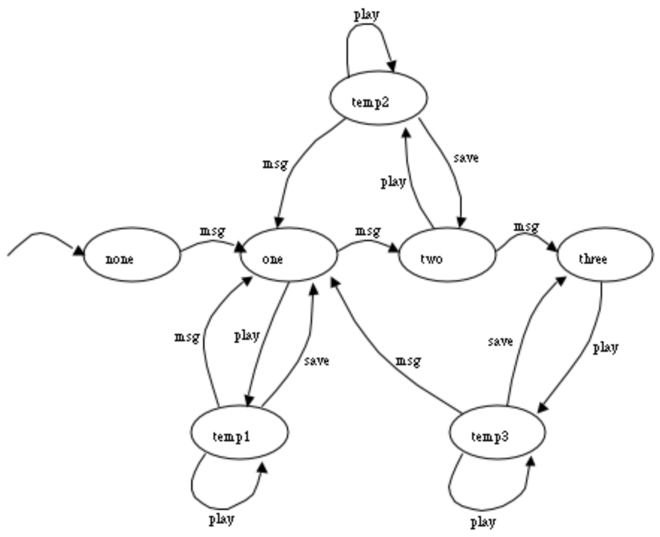
CMU 17-651 Models of Software Systems Fall 2018

# Homework #5: State Machines

Dario A Lencina-Talarico Due: 1 October 2018

1. A certain, simple, answering machine has two buttons, "play" and "save" and can, of course, receive messages. If someone plays the messages and doesn't save them, they are erased/overwritten when the next incoming message is received. The answering machine only holds a specified number of messages; when it reaches full capacity it refuses to accept new messages. The answering machine can be modeled by the state machine, **AnsMachine**, whose state transition diagram is attached.



(a) Give a 4-tuple description for this state machine.
AnsweringMachine == (
$\{none, one, two, three, temp1, temp2, temp3\}$,
$\{none\}$,
$\{msg, play, save\}$,
$\{(none, msg, one)$,
$(one, msg, two), (one, play, temp1)$,
$(two, msg, three), (two, play, temp2)$,
$(temp1, save, one), (temp1, msg, one), (temp1, play, temp1)$,
$(temp2, save, two), (temp2, play, temp2), (temp2, msg, one)$,
$(temp3, play, temp3), (temp3, save, tree), temp3, msg, one)\}$
)

(b) Give three execution fragments of **AnsMachine**, at least one of which is not an execution.
Given the GWC10 definition of execution fragment and execution:

"Definition 3. An execution fragment is a finite or infinite sequence ⟨ s0,a1,s1,a2,s3,... ⟩ of alternating states and actions such that for all i (si,ai+1,si+1) is a step of M."

"Definition 4. An execution is an execution fragment starting with an initial state of M (and ending in a state if finite)"

i. $\langle none, msg, one, play, temp1, msg, one, msg, two\rangle$

ii. $\langle none, msg, one, msg, two, play, temp2, save, two\rangle$

iii. Execution which is not an execution frament:
$\langle two, msg, three, play, temp3, play, temp3, msg, one\rangle$
It is not an execution fragment because it starts at state two as opposed to *none*

(c) Give both a finite and an infinite execution of **AnsMachine**.

i. finite == $\langle none, msg, one, play, temp2, save\rangle$

ii. infinite == $\langle none, msg, one, play, temp1, msg, one, play, temp1, ...\rangle$
I am not sure if I am using the right notation for the infinite execution, since *none* is not repeated, only the $\{msg, one, play, temp1\}$

(d) For each of the following, indicate whether or not it is an event-based trace of **AnsMachine**.
Logic used to answer: I determined if the presented sequences of actions are valid or not for AnsMachine:

i. $\langle play, save, msg, msg, play\rangle$
Not valid because the initial state none does not handle the play event.

ii. $\langle msg, msg, msg, play, save, msg, msg, msg\rangle$
Not valid because after the element 5 you end up in three which does not handle msg

iii. $\langle msg, play, save, msg, msg, play, msg, msg, msg\rangle$
It is a valid event-based trace, the final state is *three*

(e) Give two examples of state-based traces of **AnsMachine**.

i. $\langle none, one, two, three\rangle$

ii. $\langle none, one, temp1, one, two, temp2\rangle$

(f) Give two sequences of states that are not state-based traces of **AnsMachine**.

i. $\langle temp3, temp2, two\rangle$

ii. $\langle none, three, two, temp3\rangle$

(g) What are the reachable states of this machine?
The reachable states are the ones that we can get to given the set of state transition relations.
For **AnsMachine**, reachableStates == $\{none, one, two, three, temp1, temp2, temp3\}$

(h) Is **AnsMachine**'s event-based behavior finite or infinite?
It is event-based behavior infinite because we can create an infinite traces if we send the play message repeatedly while in any of the following states $\{one, twothree, temp1, temp2, temp3\}$

(i) Can a state machine $M = (S, I, A, \delta)$ with an infinite trace have finite behavior? Give an example or explain why not.

From GWC10, Chapter 9.6:

"Recall the red and blue light example where the behavior of the light is infinite because there are an infinite number of traces associated with the light. Also, some of those traces are infinite (e.g., pressing red forever).

In some models of state machines, in particular, Communicating Sequential Processes (CSP), infinite traces are not included in the machine's behavior. This kind of model is sometimes called the finite-trace model. The behavior of the machine is defined to be a possibly infinite set of finite traces.

Why is this a reasonable model? It has a simple structure which has some nice mathematical properties. Using this model may help simplify our thinking about the system; we know every trace in the behavior is finite so we do not have to worry about infinite traces. With a model that has both finite and infinite traces, whenever we prove some property about the behavior it describes, we usually have to structure our proof into two parts, one to handle the finite trace case and one to handle the infinite trace case. But perhaps the most compelling intuitive argument for this model is that we can never see an infinite execution; if we cannot observe this behavior then why should we model it?"

2. Consider a TV remote control that allows the user to select the channel to be viewed, add or remove a "parental block" to a channel that prevents the channel from being displayed (removal requires a password), and enter a password to allow a blocked channel currently selected to be displayed. If a blocked channel is selected, the channel is not initially displayed. The user may choose to select a different channel or may enter the password to display the channel. If the incorrect password is entered, the channel is not displayed. If the correct password is entered, the channel is displayed.

   Your task is to model the described functionality of the remote control. That is,

   (a) Specify the set of states

   (b) Specify the pre- and post-conditions for each action

   Your solution should satisfy the following requirements:

   i. Do *not* model any functionality other than that described above. In particular, assume that the correct password is fixed and cannot be changed.

   ii. Assume that the set of channels is $Channels == \{n : \mathbb{N} \mid 1 \leq n \leq 100\}$.

   iii. You may only use the following actions in your model:
   - *select*: Select a channel for viewing
   - *correctpw*: Enter correct password
   - *incorrectpw*: Enter incorrect password
   - *addblock*: Block selected channel
   - *removeblock*: Unblock selected channel

   iv. If (and only if) the requirements are ambiguous, state any assumptions that you made to resolve those ambiguities.

Assumptions:

(a) TV power modes are out of scope.

(b) Communication errors between the tv and the remote are out of scope.

$S = \{current\_channel : Channels, blocked : \{false, true\}\}$

select(c:Channels)
**pre** true
**post** current_channel = c

addblock
**pre**$blocked = false$
**post** blocked = true

removeblock
**pre**$blocked = true$
**post**$(correctpw \Rightarrow blocked = false) \wedge$
$(incorrectpw \Rightarrow blocked = true)$

correctpw
**pre**$true$
**post**$true$

incorrectpw
**pre**$true$
**post**$false$

In my opinion, is channel displayed can be modeled as an action:

isChannelDisplayed(c:Channel) / {true,false}
**pre**$true$
**post**$c = current\_channel \wedge \neg blocked$