

# MTAT.03.083 – Systems Modelling

## Regular Exam – 14 January 2014

### Notes:

- The exam is open-book and open-laptop. Web browsing is allowed.
- You are not allowed to communicate with anyone during the exam in any way (except with the lecturer).
- You may submit part 1 of your exam on paper, or electronically.
- You must submit part 2 of your exam electronically in the form of a pnml file generated with WoPeD.
- Include all files of your submission in a zip file and submit it using the “Submit” button in the course Web page.
- If you find that there is not enough information in the text below and you need to make additional assumptions, please write down your assumptions.

### **Part 1. An Application for Executing Statecharts**

A statechart consists of states and transitions. A transition is a directed arc from one state (the “source” state) to another state (the “target” state). Every transition is labelled with an event (in this exercise we do not consider conditions nor actions).

A state may be initial or final. An initial state is a state that has no incoming transitions, while a final state is a state that has no outgoing transitions. We assume that a statechart has only one initial state but can have multiple final states.

Any state (other than the initial or final states) can be simple or compound. A compound state contains one or multiple statecharts under it. When a compound state contains one single statechart it is called an OR state, whereas if it contains multiple statecharts, it is called an AND state. An example of a statechart is given in Figure 1. In this figure, we assign numerical identifiers to states, but you are not required to model these numerical identifiers since they are not necessary for execution purposes.

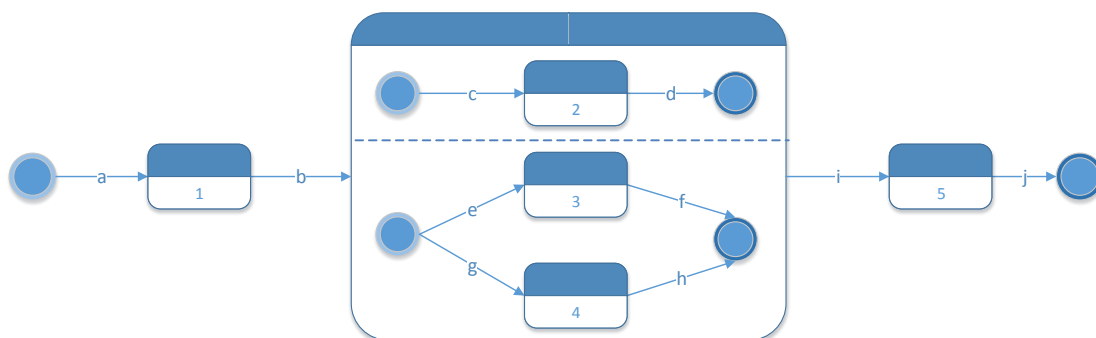


Figure 1. Sample statechart with an AND-state

An execution of a statechart starts in the initial state. The execution moves from one state to another by consuming an event. An event E can be successfully consumed if the current state of the statechart’s execution has an outgoing transition T that is labelled with event E. Otherwise, if the current state does not have any outgoing transition labelled with event E, then event E cannot be consumed. When an event E is successfully consumed, the statechart’s execution moves to the target state of T. For example, if an event “b” is received when the above statechart is in state 1,

then the event is successfully consumed and the execution moves to the AND-state. On the other hand, if an event “f” is received when the above statechart is in state number 1, then the event cannot be consumed.

A sequence of events (e.g., [a, b, d, e, c]) – herewith referred to as a *word* – is accepted by statechart M if there is an execution of statechart M that can successfully consume every event in the word one after the other, and after consuming the last event in the word, the execution is in a final state of M. If after consuming the last event in the word, the execution is in a non-final state, we say that the word is *partially accepted*. Otherwise, we say that the sequence is *not accepted*.

When an execution moves to a compound state, an execution of each of the statecharts under this compound state is created and started (remember that the execution of a statechart starts in its initial state). When a statechart’s execution is in a compound state C, an event E can be successfully consumed if either:

- The state C itself has an outgoing transition T labelled with E, in which case the execution of the statechart moves to the target of T. For example, event “i” can be successfully consumed when an execution of the statechart in Figure 1 is in the AND-state. In this case the statechart’s execution moves to state 5.
- Any of the executions of the statecharts directly contained in state C can successfully consume event E. In which case the state of the execution of the contained statechart is updated accordingly. For example, event “e” can be successfully consumed when the execution of the statechart of Figure 1 is in the initial state of the lower statechart of the AND-state. In this case the execution of this lower statechart moves to state 3.

For simplicity, we represent an event as a single alphabetic character (e.g., ‘a’, ‘b’, etc.) while words are represented as strings. For example, the word [a, b, d, e, c] is represented as “abdec”.

For example, the statechart in Figure 1 accepts the following words:

- abcefdij
- abcgdhij
- abcij
- abij

The same statechart partially accepts the following words:

- ab
- abe

And it does not accept the following words:

- abcdegij
- abdcfeij

The aim of this question is to design an application that takes as input a statechart and a word, and determines whether the input word is accepted, partially accepted or not accepted by the given statechart.

## Tasks

1. For each of the following words, indicate if the statechart in Figure 1 accepts it, partially accepts it, or does not accept it: **[3 points]**

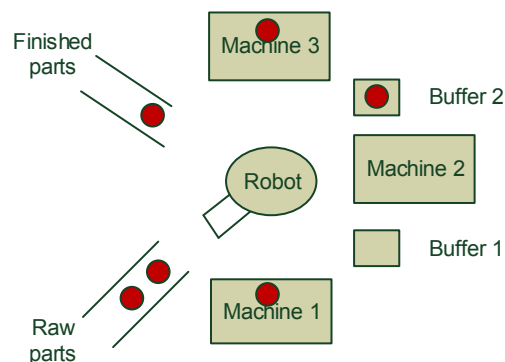
- abecfi
  - abcfdij
  - abcgdhij
2. Design a domain model (UML class diagram) of statecharts and their executions as defined above. **[8 points]**
  3. Write a sequence diagram that describes how the classes in the domain model (and other classes if needed) interact in order to determine if a given event can be successfully consumed by an execution of a statechart. Assume that the event is produced by an object of an external class called EventFeeder. **[8 points]**
  4. Write a sequence diagram that describes how the classes in the domain model (and other classes if needed) interact in order to determine if a given input word is accepted, partially accepted or not accepted. Assume that the word is produced by an object of an external class called WordFeeder. **[8 points]**
  5. Design an application model for the above scenario. The application model should include methods required to determine if a statechart execution can consume or not a given input event, as well as methods required to determine if a statechart accepts, partially accepts or does not accept a word. **[8 points]**

## **Part 2. A Pipelined Factory**

A section of a pipelined factory (cf. Figure 2) is composed of three machines, one robot and two buffers. A robot moves parts from raw line to M1, from M1 to M2, from M2 to M3 and finally from M3 to finished line. M1 and M2 can hold maximum 1 part. If M2 is not empty, it is possible to move a part from M1 to buffer1. If M3 is not empty, it is possible to move a part from M2 to buffer2. Buffer1 and buffer2 can hold maximum 1 part. At any point in time, moving parts to M3 has higher priority than moving parts to M2. Therefore, a part can be moved to M2 only if there is no part in buffer2 and there is no part in M2.

### **Task**

6. Model the pipelined factory with a plain Petri net. Add tokens to the Petri net to represent the configuration shown in the figure. **[15 points]**



**Figure 2. Section of a pipelined factory**