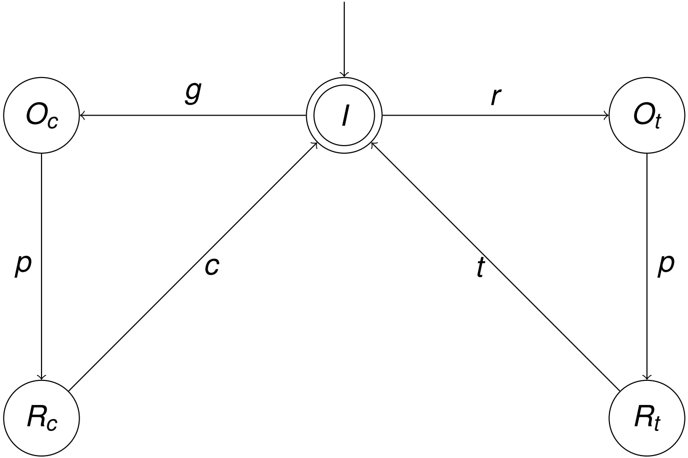
Embedded systems specification and design

David Kendall

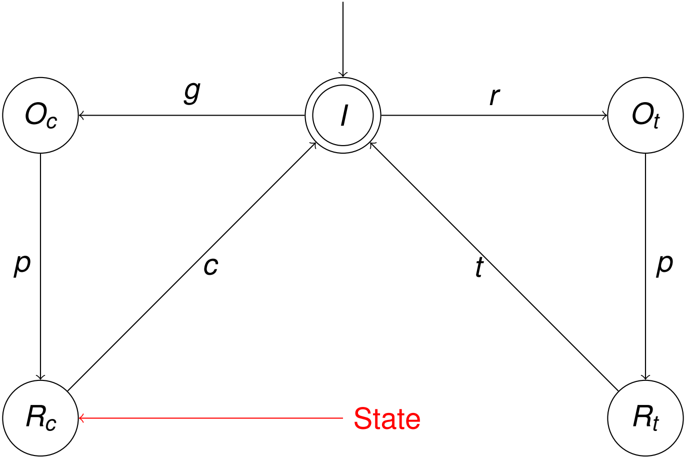
Introduction

- Finite state machines (FSM)
- FSMs and Labelled Transition Systems
- FSMs and Formal Languages
- Modelling a system as a FSM
- Specifying system properties with FSMs

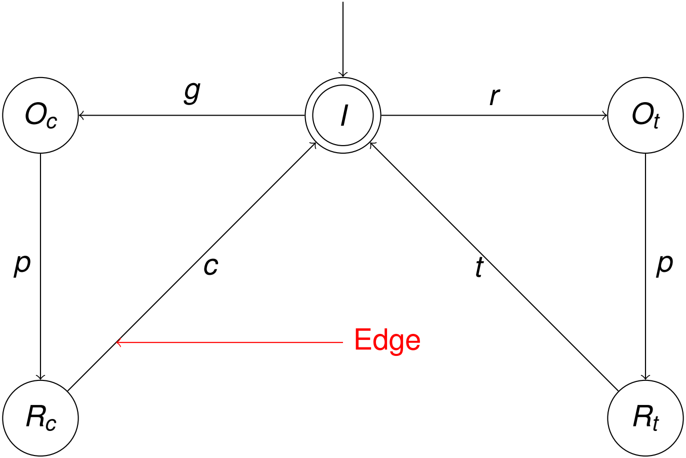
Finite State Machine (FSM)



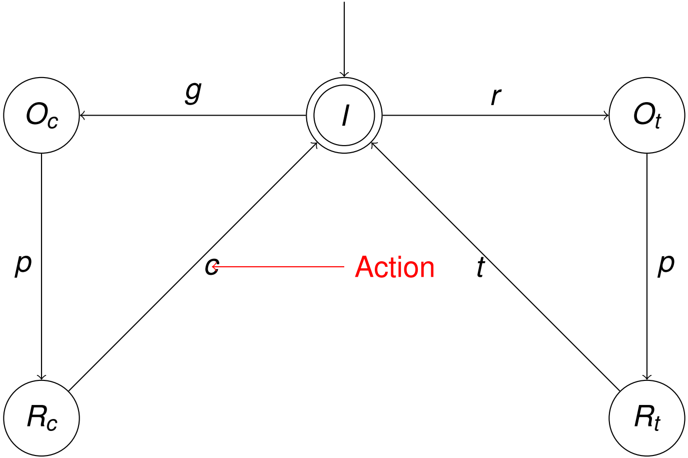
Finite State Machine (FSM)



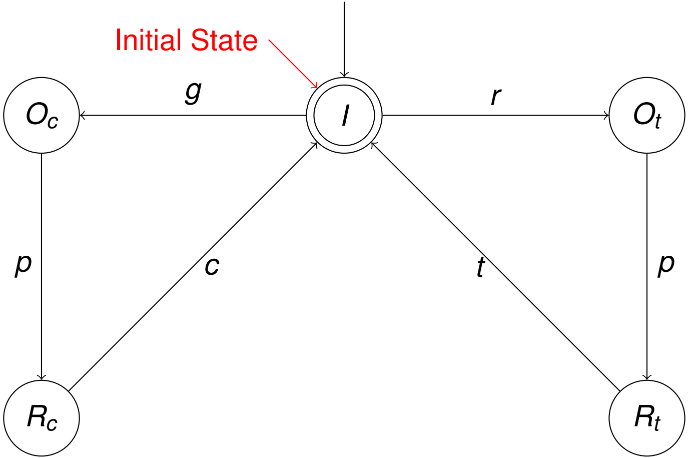
Finite State Machine (FSM)



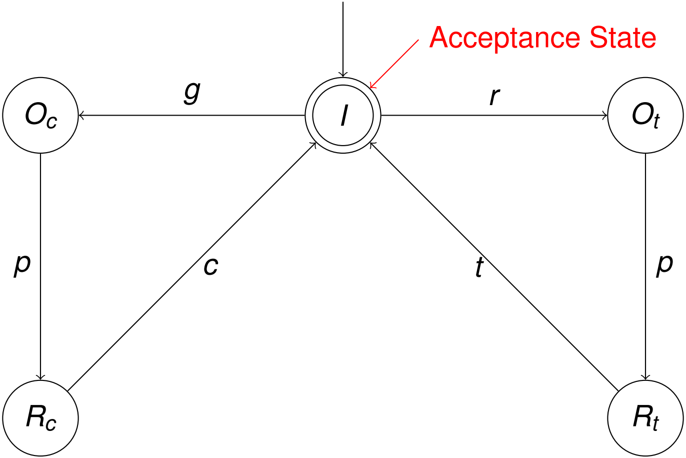
Finite State Machine (FSM)



Finite State Machine (FSM)



Finite State Machine (FSM)



Finite State Machine (FSM)

- Finite State Machines (FSM) are also known as Finite State Automata (FSA). We'll use these names to mean the same thing.
- Formally, a FSM is a tuple $M = (S, s^0, A, E, F)$ where
- S is a set of states
- s^0 is a distinct state called the initial state
- A is an alphabet (also called a set of actions or labels or symbols)
- $E \subseteq S \times A \times S$ is a set of edges
- $F \subseteq S$ is a set of acceptance states (also called final states)

Finite State Machine (FSM)

- Write down the formal statement of the example FSM

Finite State Machine (FSM)

- Write down the formal statement of the example FSM
- $S = \{I, O_c, O_t, R_c, R_t\}$

Finite State Machine (FSM)

- Write down the formal statement of the example FSM
- $S = \{I, O_c, O_t, R_c, R_t\}$
- $s^0 = I$

- Write down the formal statement of the example FSM
- $S = \{I, O_c, O_t, R_c, R_t\}$
- $s^0 = I$
- $A = \{c, g, p, r, t\}$

Finite State Machine (FSM)

- Write down the formal statement of the example FSM
- $S = \{I, O_c, O_t, R_c, R_t\}$
- $s^0 = I$
- $A = \{c, g, p, r, t\}$
- $E = \{(I, g, O_c), (I, r, O_t), (O_c, p, R_c), (O_t, p, R_t), (R_c, c, I), (R_t, t, I)\}$

Finite State Machine (FSM)

- Write down the formal statement of the example FSM
- $S = \{I, O_c, O_t, R_c, R_t\}$
- $s^0 = I$
- $A = \{c, g, p, r, t\}$
- $E = \{(I, g, O_c), (I, r, O_t), (O_c, p, R_c), (O_t, p, R_t), (R_c, c, I), (R_t, t, I)\}$
- $F = \{I\}$

Labelled Transition System

- Sometimes, we omit the acceptance states F from the statement of a FSM – in this case, we assume that $F = S$, i.e. all the states are acceptance states
- Such a FSM corresponds directly to a labelled transition system (LTS)
- A different notation for a LTS may be used where
  - the alphabet is called the set of labels and is noted $\mathcal{L}$
  - the set of edges is called the transition relation and is noted $\longrightarrow$
  - if $(s, a, s') \in \longrightarrow$ we may write $s \xrightarrow{a} s'$
- So, a LTS is a tuple $(S, s^0, \mathcal{L}, \longrightarrow)$ where the components correspond to the FSM components, with notational changes as above

Labelled Transition System

- An execution or run or trace of a LTS is a finite or infinite sequence of alternating states and labels

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \ldots$$

where $s_i \xrightarrow{a_i} s_{i+1}$, i.e. the sequence respects the transition relation of the LTS

Formal Language

- Given an alphabet, A, a string (or word) over A is a sequence of zero or more symbols $a_0 a_1 \ldots a_n$ where each $a_i \in A$
- There is only one zero-length string, called the empty string, usually written as ϵ
- A formal language is a set of strings over some alphabet A
- The notation A^n is used for the set of all strings of length n over A
- The notation A^* is used for the set of all strings over A, i.e. $A^0 \cup A^1 \cup A^2 \cup A^3 \ldots$

Formal Language Exercise

- Make up some alphabets and check understanding of the notation.

Running a FSM to accept a string

- A FSM $M = (S, s^0, A, E, F)$ accepts (recognises) a set of strings (a language)
- To determine if M accepts a string $w = a_0 a_1 \ldots a_n$, construct a run of M as follows:
  - Start in the initial state s^0 and consider the symbol a_0
  - If M is in a state s, considering symbol a, and $s \xrightarrow{a} t$, move M into state t and consume a; repeat
  - If a run consumes all symbols in the string w and terminates with M in an acceptance state, then the run accepts w
- If any run of M accepts w, then M accepts w

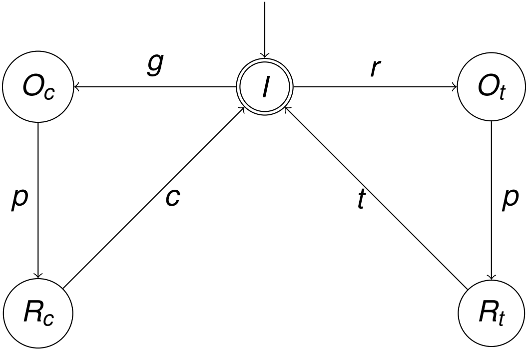
The Language of a FSM

- The language of M is written $L(M)$ and is the set of all strings accepted by M. Notice that $L(M) \subseteq A^*$
- $a_0 a_1 a_2 \ldots a_{n-1} \in L(M)$ iff there is a run

  $$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots \xrightarrow{a_{n-1}} s_n$$

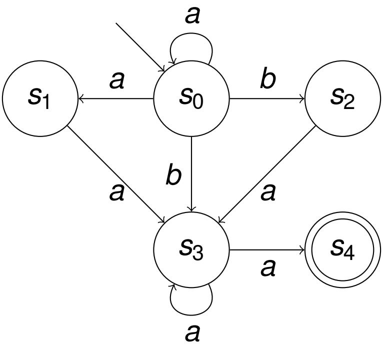
  where $s_0 = s^0$ and $s_n \in F$.

FSM Language Exercise



- Work through the recognition process with this machine and some example strings

Finite State Machine (FSM)



- Write down the formal statement of this FSM
- Check if the following strings are accepted by this FSM: (1) *aba*, (2) *aaba*, (3) *aaaba*, (4) *abb*, (5) *a*, (6) *aaa*, (7) *ba*

Deterministic and Complete FSM

Definition (Deterministic FSM)

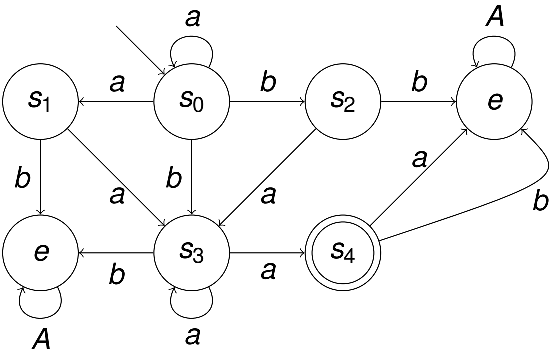
A FSM $M = (S, s^0, A, E, F)$ is deterministic if for every $s \in S$ and every $a \in A$ there is at most one $s' \in S$ such that $(s, a, s') \in E$

Definition (Complete FSM)

A FSM $M = (S, s^0, A, E, F)$ is complete if for every $s \in S$ and every $a \in A$ there is at least one $s' \in S$ such that $(s, a, s') \in E$

- The previous example is not deterministic (i.e. it is non-deterministic) and is not complete

A Complete FSM



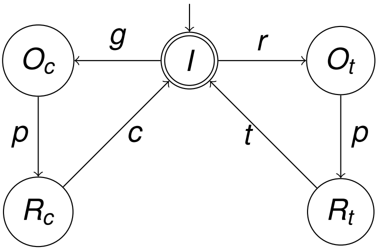
Notes on a complete FSM

- A complete FSM can never ‘get stuck’ when run on an input string – the string will be consumed and the machine will halt
- An incomplete FSM can be made complete by adding an additional state to act as the target of all missing edges
- e acts as a sink (or error) state, i.e. any missing edges are directed to e (duplicated to reduce clutter in the diagram)
- Labelling an edge with a set such as A is a shorthand for several edges, each labelled with a label from the set.
- Sometimes, the notation $A \setminus B$ is used to mean the set of all labels in A except those that are also in B, e.g. $A \setminus \{a\} = \{b\}$

Modelling an embedded system as a FSM

- An embedded system can be modelled using a FSM
- States of the FSM are the states of the system
- Alphabet of the FSM models the atomic actions of the system
- Edges of the FSM show what actions are possible in any given state and what state change is caused when the action is taken
- A string of the FSM gives a possible sequence of actions of the system

FSM Modelling Example



- This FSM can model a drinks machine
- I – idle, O_c – coffee ordered, O_t – tea ordered, R_c – coffee ready, R_t – tea ready
- g – press green button, r – press red button, p – prepare drink, c – fill cup with coffee, t – fill cup with tea
- $I \xrightarrow{g} O_c$ means when the system is idle, the green button can be pressed and the system enters the coffee ordered state. The word *gpcrpt* is a possible run of the system

Embedded systems do not terminate

- Embedded systems are assumed to interact continuously with their environment. Their executions are regarded as non-terminating
- How does this fit with our idea of the language of a FSM and the acceptance of strings?

Embedded systems do not terminate

- Embedded systems are assumed to interact continuously with their environment. Their executions are regarded as non-terminating
- How does this fit with our idea of the language of a FSM and the acceptance of strings? It doesn't.

Embedded systems do not terminate

- Embedded systems are assumed to interact continuously with their environment. Their executions are regarded as non-terminating
- How does this fit with our idea of the language of a FSM and the acceptance of strings? It doesn't.

Definition (Büchi Acceptance)

An infinite word w (execution) is accepted by a FSM M if there is some run of M over w that visits an acceptance state infinitely often.

Specifying system properties

- Given a model of a system, we would like to check that it behaves properly, i.e. that it only does what it should do
- For the drinks machine, we might want to check that
  - It does not dispense a drink unless a button has been pressed
  - If the green button is pressed, eventually a cup of coffee is produced
  - If the red button is pressed, eventually a cup of tea is produced
  - ...
- How do we specify the properties of the system that are permissible? Use another FSM.

Checking system properties

- Let M_{sys} be the FSM that models the actual behaviour of the system.
- Let M_{spec} be a FSM that specifies permissible behaviour of the system.
- Then, to check that the system satisfies its specification, we check that
  - $L(M_{sys}) \subseteq L(M_{spec})$
  - i.e. all actual behaviours are permissible
- We will see how this can be automated next.