Programming Languages Question

March 13, 2000

We are going to look at a programming language called UPPAAL that is used to describes real-time systems based on finite state automata.

1 Part 1 (70 points)

Programs in UPPAAL in this simplified language describe ordinary finite state automata (that you study in CS450) over a given alphabet Σ. The following syntax, shown as a BNF, is adopted by UPPAAL to describe an individual automata

$$
\begin{array}{lcl}
\textbf{Automata} & \rightarrow & \textit{process}\ \textbf{Id}\{\textbf{States Initial Transitions}\} \\
\\
\textbf{States} & \rightarrow & \textit{state}\ \textbf{StateList}\ ; \\
\\
\textbf{StateList} & \rightarrow & \textbf{State} \\
 & | & \textbf{State}, \textbf{StateList} \\
\\
\textbf{State} & \rightarrow & \textbf{Id} \\
\\
\textbf{Initial} & \rightarrow & \textit{init}\ \textbf{State}\ ; \\
\\
\textbf{Transitions} & \rightarrow & \textit{trans}\ \textbf{TransList} \\
\\
\textbf{TransList} & \rightarrow & \textbf{Transition}\ , \textbf{TransList} \\
 & | & \textbf{Transition} \\
\\
\textbf{Transition} & \rightarrow & \textbf{State}\ \mapsto\ \textbf{State}\ \{\textbf{Symbol}\}
\end{array}
$$

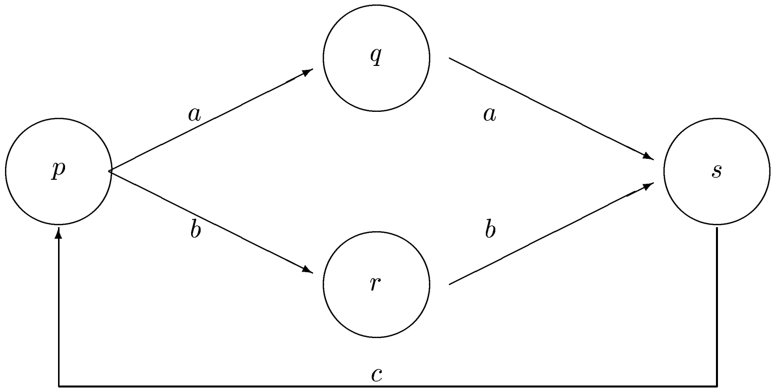
An execution is obtained by supplying a string (composed by symbols from the alphabet Σ) to the program; the program will process the string by applying the automata transitions (starting from the initial state) and produce as result the final state reached after processing the string. If the execution enters a state q and there are no transitions leaving q labeled with the current input symbol, then the computation will immediately stop and return the value *failure*. In any other case the computation will be considered successful.

For simplicity you can assume that for each state q in the automata and for each symbol a in Σ, there is at most one transition labeled a and starting from state q. Observe also that in this definition of finite state automata there are not *final* states—a string is accepted as long as there is a path in the automata labeled with the input string.

For example, the following UPPAAL program

```
process automata1
  {
      state p, q, r, s ;
      init  p;
      trans p -> q {a} ,
            p -> r {b} ,
            q -> s {a} ,
            r -> s {b} ,
            s -> p {c}
  }
```

implements the following automata:



The program will produce successful computations for each substring of the language $((aa + bb)c)*$.

Provide the denotational semantics for this language—i.e., provide an accurate description of the semantic domains needed and formally describe the valuation functions (both their signatures as well as their definitions).

2 Part 2 (30 points)

An actual real-time system is typically composed by multiple components, each described by a separate automaton. The different automata operate on the same input timed string, and the final result is the described by the final state reached in each automata after consuming the string. The different automata work in lock-step by synchronously consuming symbols of the input string. E.g., at the first step all automata perform a transition driven by the first symbol in the string, then they all perform a transition based on the second symbol of the string, and so on.

Precisely describe how to extend syntax and semantics of UPPAAL to accommodate for multiple automata operating in a synchronous fashion and consuming the same input string.

Solution

Solution Part 1

The denotational semantics is composed by the syntax given in the problem, a collection of domains, and the valuation rules which associate an element of a domain to each syntax construction.

The domains used are those needed to model the behavior of a finite state automata. We can use Id to represent the states of the automata ($Q = Id$), while we assume that there is a given set Σ representing the alphabet used. Each transition can be represented as a function which maps pairs $\langle n, a\rangle \in Q \times \Sigma$ to a new state from Q.

The meaning of the whole automata is a function which takes a string and produces an element of $Q_f = Q \cup \{fail\}$.

The sketch of the valuation functions is the following:

- the top function:

  $$\begin{array}{ll} \textbf{Automata}[\text{process I \{ St In Tr \}}] = & \lambda s. \\ & \text{let states=}\textbf{States}[\text{St}] \text{ in} \\ & \text{let in} = \textbf{Initial}[\text{In}] \text{ in} \\ & \text{let fun} = \textbf{Transitions}[\text{Tr}] \text{ in} \\ & \text{(fun s in)} \end{array}$$

- **States**[state List] = **StateList**[List]($\lambda x.fail$)

- **StateList**[Id] = $\lambda f.[Id \mapsto Id]f$ and **StateList**[Id,List] = $\lambda f.[Id \mapsto Id]$**StateList**[List]$f$

- **Initial**[init Id] = Id

- **Transitions**[trans List] = let fun = **TransList**[List]($\lambda\, s.\lambda\, a.fail$) in (processstring fun)

  where the processstring is defined as follows:

  $$\begin{array}{ll} \mathit{processstring} & = \lambda fun.\lambda str.\underline{\lambda} in. \\ & \mathit{empty}(str) \mapsto in \mid \mathit{processstring}(fun, (cdr\ \ str), (fun\ \ in)) \end{array}$$

  Note that the function is strict on the in argument. The strictness is associated to the special value **fail** of the state domain. Thus, as soon as a **fail** is produced, it is immediately propagated to the rest of the computation.

- the transitions are functions that takes as input a state and a symbol and return a new state.

  $$\begin{array}{ll} \textbf{TransList}[Id_1 \mapsto Id_2\{Id_3\}] = & \lambda\, fun. \\ & \lambda\, s.\lambda\, a.(Id_1 = s \wedge Id_3 = a \mapsto Id_2 \mid (fun\, s\, a) \end{array}$$

$$\mathbf{TransList}[Id_1 \mapsto Id_2\{Id_3\}, List] = \begin{array}{l} \lambda\, fun. \\ \lambda\, s. \lambda\, a. (Id_1 = s \wedge Id_3 = a) \mapsto Id_2 \mid \mathbf{TransList}[List](fun, s, a) \end{array}$$

Part 2

Since we have multiple automata, the state of the computation is now a tuple which contains the state of each automata. The valuation functions for the automata i will provide a transition function f_i. One step of computation starting from state $\langle s_1, \ldots, s_n \rangle$ and consuming symbol a will produce the resulting state $\langle f_1(s_1), \ldots, f_n(s_n) \rangle$. The only additional problem one should consider is the possibility that $f_i(s_i)$ may produce **fail**. In this case the whole state should be mapped to **fail**. Note that **fail** can be simply treated as $\perp$ and the valuation function should be strict.