# Finite State Machines: Definitions; Verification

Greg Plaxton
Theory in Programming Practice, Spring 2005
Department of Computer Science
University of Texas at Austin

# Finite State Machine: Definition

- A (deterministic) finite state machine consists of:
  - A *finite* number of states, where one state is designated as the *initial* state, and a subset of the states are designated as *accepting*
  - A *state transition function* that specifies the next state for each state and input symbol
- A finite state machine *accepts* or *rejects* each *finite* string over the input alphabet
- To determine whether a given finite string is accepted, start in the initial state and repeatedly update the state according to the next input symbol and the state transition function
  - The input string is accepted if and only if this process terminates in an accepting state

# Finite State Machine: Correspondence to a "Language"

- Let $A$ denote the input alphabet of some FSM $M$
- As we have seen, $M$ accepts some subset of the set of all finite strings over $A$
- The set of strings accepted by $M$ is the *language* accepted by $M$

# Finite State Machine: Pictorial Representation

- Each state is depicted by a circle
- An arrow labeled "start" points to the initial state
- Each accepting state has a double circle
- The transition function is specified by drawing arrows labeled by either a single input symbol or a set of input symbols
  - The label on an arrow associated with a transition from state $u$ to state $v$ indicates which input symbols cause this transition to be taken
  - Normally, a transition is explicitly specified for each state and input symbol
  - Remark: If certain transitions are not explicitly specified, the intended interpretation is that any input string leading to an unspecified transition is rejected

# Verification of FSMs

- In a previous lecture we designed a few simple FSMs that seemed to be correct in terms of accepting a specified language
  - Example: The finite state machine we designed to accept words containing the five vowels in order
- How can we formally verify the correctness of such FSMs?
  - Our strategy is to label each state with (our guess as to) the set of finite strings leading to that state
  - These labels may be verified using induction (assuming that our guesses are correct)

# Verification Procedure

- First, annotate each state with a predicate over finite strings
  - The predicate defines a set of input strings, namely, those for which the predicate holds (i.e., evaluates to true)
  - This corresponds to our guess as to the set of input strings leading to this state
  - As such, the sets of input strings defined by the annotations is required to partition the set of all input strings
- Second, show that the predicate associated with the initial state holds for the empty string
- Third, prove that for any transition from a state $u$ to a state $v$ on input symbol $a$, if some finite string $x$ satisfies the state $u$ annotation, then the finite string $xa$ satisfies the state $v$ annotation

# Verification Procedure: Why Does it Work?

- Let $x$ be an arbitrary input string
- Let the given FSM $M$ be in state $u$ after processing $x$
  - Because $M$ is deterministic, the state $u$ is uniquely defined
- Let $v$ denote the unique state for which the associated annotation holds for $x$
  - The existence/uniqueness of $v$ follows from the requirement that the state annotations partition the set of all input strings
- We prove that $u = v$ by induction on the length of $x$
  - The second part of the verification procedure handles the base case
  - The third part handles the induction step

# Verification Example: Parity

- Design an FSM to accept all finite binary strings with an odd number of 0s and an odd number of 1s
- Verify the correctness of your FSM

# Verification Example: Ascending Digits

- Design an FSM to accept any finite string of decimal digits in which each successive digit is strictly higher than the preceding one (e.g., 038 or 13579)
- Verify the correctness of your FSM

# Another Example

- Design an FSM to accept all finite binary strings with an equal number of zeros and ones
- Is this possible?

# Some Closure Properties of FSMs

- Let FSMs $M_1$ and $M_2$ accept the languages $L_1$ and $L_2$, respectively
- Is it possible to give a general procedure to construct an FSM accepting $L_1 \cup L_2$ from FSMs $M_1$ and $M_2$?
- What about $L_1 \cap L_2$?
- What about $\overline{L_1}$?