LING/C SC/PSYC 438/538

Lecture 19

Sandiway Fong

Today's Topics

- **Reminder**: Homework 10 is out
- Some regular language properties
- Turing Machines (*a brief digression – similar to FSA but with a tape*)

Regular Languages and FSA

- Recall languages are sets of strings.
- Regular languages:

  1. $\emptyset$ is a regular language
  2. $\forall a \in \Sigma \cup \epsilon$, $\{a\}$ is a regular language
  3. If L_1 and L_2 are regular languages, then so are:
     (a) $L_1 \cdot L_2 = \{xy | x \in L_1, y \in L_2\}$, the **concatenation** of L_1 and L_2
     (b) $L_1 \cup L_2$, the **union or disjunction** of L_1 and L_2
     (c) L_1^*, the **Kleene closure** of L_1

- Correspondence between Regular Languages and regex devices:
  - concatenation (*juxtaposition*)
  - union (| also [ ])
  - Kleene closure (*) Note: $x^+ = xx^*$)
- **Note**:
  - backreferences **are memory devices** and thus are too powerful
  - e.g. L = {ww} and prime number testing (*see earlier lectures*)

Regular Languages and FSA

- **Closure properties:**
  - i.e. *do we still have a regular language after applying the operation*?

intersection	if L_1 and L_2 are regular languages, then so is $L_1 \cap L_2$, the language consisting of the set of strings that are in both L_1 and L_2.
difference	if L_1 and L_2 are regular languages, then so is $L_1 - L_2$, the language consisting of the set of strings that are in L_1 but not L_2.
complementation	If L_1 is a regular language, then so is $\Sigma^* - L_1$, the set of all possible strings that aren't in L_1.
reversal	If L_1 is a regular language, then so is L_1^R, the language consisting of the set of reversals of all the strings in L_1.

- Closure properties not necessarily preserved higher up *as we'll see later, e.g. context-free grammars*

Regular Languages and FSA

Textbook gives one direction only

- there are three cases:
    a) Empty string
    b) Empty set
    c) Any character from the alphabet

1. $\emptyset$ is a regular language
2. $\forall a \in \Sigma \cup \epsilon$, $\{a\}$ is a regular language

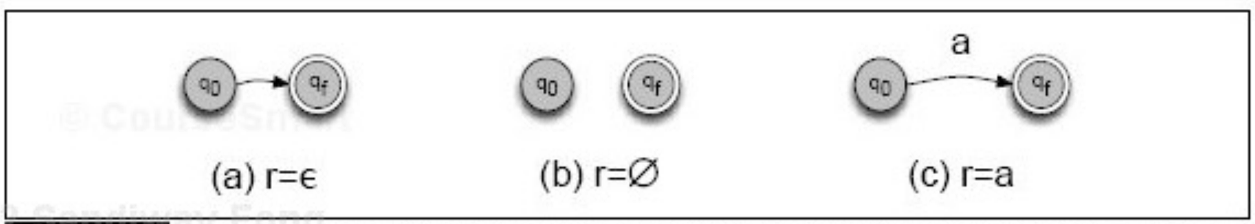


Figure 2.22 Automata for the base case (no operators) for the induction showing that any regular expression can be turned into an equivalent automaton.

Regular Languages and FSA

- **Concatenation**:

3. If L_1 and L_2 are regular languages, then so are:
   (a) $L_1 \cdot L_2 = \{xy | x \in L_1, y \in L_2\}$, the **concatenation** of L_1 and L_2
   (b) $L_1 \cup L_2$, the **union** or **disjunction** of L_1 and L_2
   (c) L_1^*, the **Kleene closure** of L_1

  - Link final state of FSA_1 to initial state of FSA_2 using an empty transition

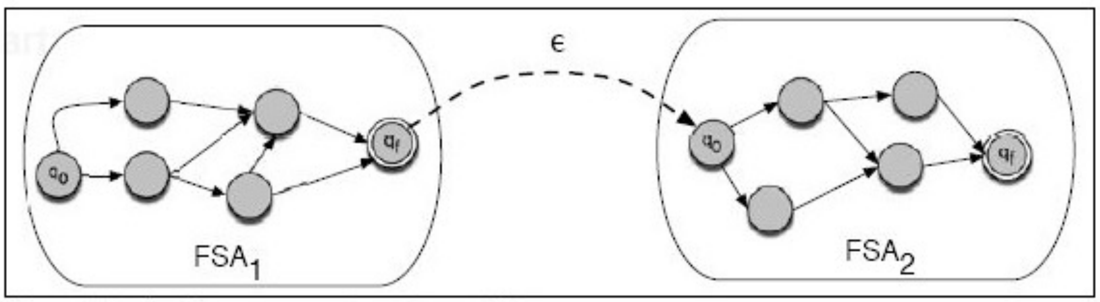


Figure 2.23 The concatenation of two FSAs.

Note: empty transition ε can be deleted using the set of states construction

Regular Languages and FSA

- **Kleene closure**:

3. If L_1 and L_2 are regular languages, then so are:
   (a) $L_1 \cdot L_2 = \{xy | x \in L_1, y \in L_2\}$, the **concatenation** of L_1 and L_2
   (b) $L_1 \cup L_2$, the **union** or **disjunction** of L_1 and L_2
   (c) L_1^*, the **Kleene closure** of L_1

  - repetition operator: zero or more times
  - use empty transitions for loopback and bypass

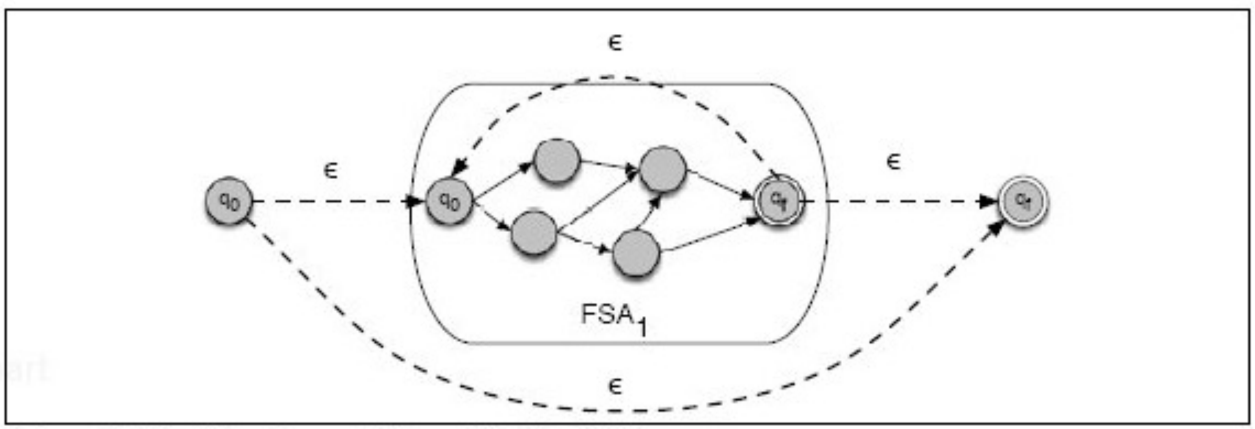


Figure 2.24 The closure (Kleene *) of an FSA.

Regular Languages and FSA

- **Union**: aka disjunction

3. If L_1 and L_2 are regular languages, then so are:
   (a) $L_1 \cdot L_2 = \{xy | x \in L_1, y \in L_2\}$, the **concatenation** of L_1 and L_2
   (b) $L_1 \cup L_2$, the **union or disjunction** of L_1 and L_2
   (c) L_1^*, the **Kleene closure** of L_1

  - Non-deterministically run both FSAs at the same time, accept if either one accepts

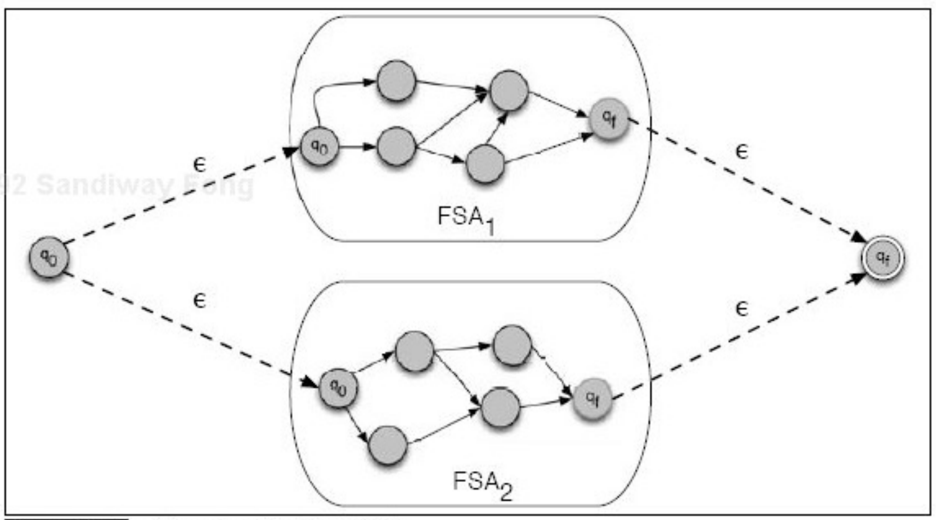


Figure 2.25 The union (|) of two FSAs.

Regular Languages and FSA

- Other closure properties:

intersection	if L_1 and L_2 are regular languages, then so is $L_1 \cap L_2$, the language consisting of the set of strings that are in both L_1 and L_2.
difference	if L_1 and L_2 are regular languages, then so is $L_1 - L_2$, the language consisting of the set of strings that are in L_1 but not L_2.
complementation	If L_1 is a regular language, then so is $\Sigma^* - L_1$, the set of all possible strings that aren't in L_1.
reversal	If L_1 is a regular language, then so is L_1^R, the language consisting of the set of reversals of all the strings in L_1.

Let's consider building the FSA machinery for each of these guys in turn…

Regular Languages and FSA

- Other closure properties:

intersection	if L_1 and L_2 are regular languages, then so is $L_1 \cap L_2$, the language consisting of the set of strings that are in both L_1 and L_2.

- What would be the final state?

Regular Languages and FSA

- Other closure properties:

difference	if L_1 and L_2 are regular languages, then so is $L_1 - L_2$, the language consisting of the set of strings that are in L_1 but not L_2.

- What would be the final state?

Regular Languages and FSA

- Other closure properties:

complementation	If L_1 is a regular language, then so is $\Sigma^* - L_1$, the set of all possible strings that aren't in L_1.

Example: $\Sigma = \{a,b\}$; $\Sigma^* = \{a, b\}^*$

- we need explicit arcs for each character in Σ
- then flip accepting and non-accepting states

Regular Languages and FSA

- Other closure properties:

reversal	If L_1 is a regular language, then so is L_1^R, the language consisting of the set of reversals of all the strings in L_1.

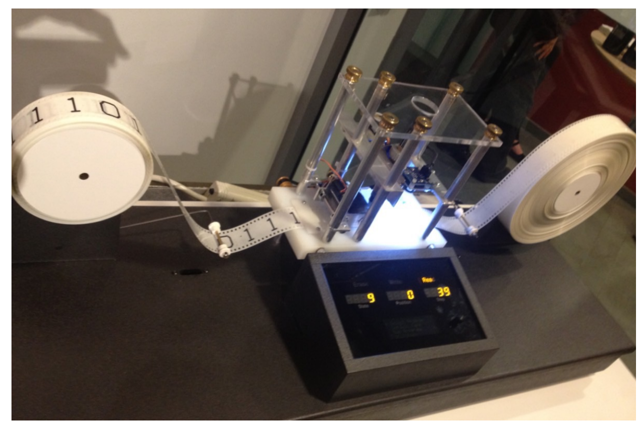
reverse arrows and swap initial/final

Turing Machine

- TM = (Q, Σ, q_0, δ) + tape
- Q, Σ finite. ∴ δ finite.
- (partial) transition function δ:
  - $Q \times \Sigma \rightarrow Q \times \Sigma \times d$ $\quad$ d = [LRN]
  - ⟨q, σ, q', σ', d⟩
- Memory:
  - one-way infinite tape
  - initially: ▷ *input* …, head at start of input
  - ▷ is the end-of-tape symbol (part of Σ)
  - how to signal right end? Suppose 0. Σ = {▷, 0} + new symbols (minimum 1)
  - initial values of cells beyond the input = 0
  - assume cannot fall off the left side
- Halt:
  - when no matching transition exists

- Encodable as a (finite) sequence of numbers:
  - ⟨|Q|, Q, |Σ|, Σ, q_0, q, σ, q', σ', d,… ⟩
- Configuration:
  - ⟨Tape, position, q⟩
  - initially: ⟨ ▷·input, 1, q⟩

FSA +



Model of a basic Turing machine, part of the Go Ask Alice exhibit at the Harvard Collection of Historical Scientific Instruments.

Turing Machine

It can be configured as a decider like an ordinary FSA

- Or it can be a transducer (i.e. *map input into output strings*)

Example

- Successor function:
  - $\Sigma = \{\triangleright, 1, 0\}$, $Q = \{q_0, q_1\}$, q_0 initial state
  - recall unary notation (initially, at the leftmost 1): ▷ 1 1 1 0 …
  - **Idea**: find 1st zero, change it to a 1
  - $\langle q_0, 1, q_0, 1, R\rangle$
  - $\langle q_0, 0, q_1, 1, R\rangle$

Turing Machine

- +2 (add 2) function:
  - $\Sigma = \{\triangleright, 1, 0\}$, $Q = \{q_0, q_1, q_2\}$, q_0 initial state
  - unary notation (initially, at the leftmost 1): $\triangleright$ 1 1 1 0 …
  - Idea: find 2 zeros, change them to 1's
  - $\langle q_0, 1, q_0, 1, R \rangle$
  - $\langle q_0, 0, q_1, 1, R \rangle$
  - $\langle q_1, 0, q_2, 1, R \rangle$

Turing Machine

Example:

- **doubler**
  1. ▷ 1 1 1 ▷ (start)
  2. ▷ 2 1 1 ▷ (change 1 to 2, move R)
  3. ▷ 2 2 1 ▷ (change 1 to 2, move R)
  4. ▷ 2 2 2 ▷ (change 1 to 2, move R)
  5. ▷ 2 2 2 ▷ (move L at ▷)
  6. ▷ 2 2 1 ▷ (change 2 to 1, move R)
  7. ▷ 2 2 1 1 ▷ (change ▷ to 1, move L)
  8. ▷ 2 2 1 1 ▷ (skip L 1's)
  9. ▷ 2 1 1 1 ▷ (change 2 to 1, move R)
  10. ▷ 2 1 1 1 ▷ (skip R L's)
  11. ▷ 2 1 1 1 1 ▷ (change ▷ to 1, move L)
  12. ▷ 2 1 1 1 1 ▷ (skip L 1's)
  13. ▷ 1 1 1 1 1 ▷ (change 2 to 1, move R)
  14. ▷ 1 1 1 1 1 ▷ (skip R L's)
  15. ▷ 1 1 1 1 1 1 ▷ (change ▷ to 1, move L)
  16. ▷ 1 1 1 1 1 1 ▷ (skip L 1's)
  17. halt

Turing Machine

Machine:

- has 4 states: {0, 1, 2, 3}
- Σ = {1, 2, ▷}
- q_0 = 0
- f = 3
- deterministic
  - transition function
  - each arc at states labeled
  - `sym,sym',Move`
  - `sym` ∈ Σ, Move ∈ `{L,R}`