LING/C SC/PSYC 438/538

Lecture 21

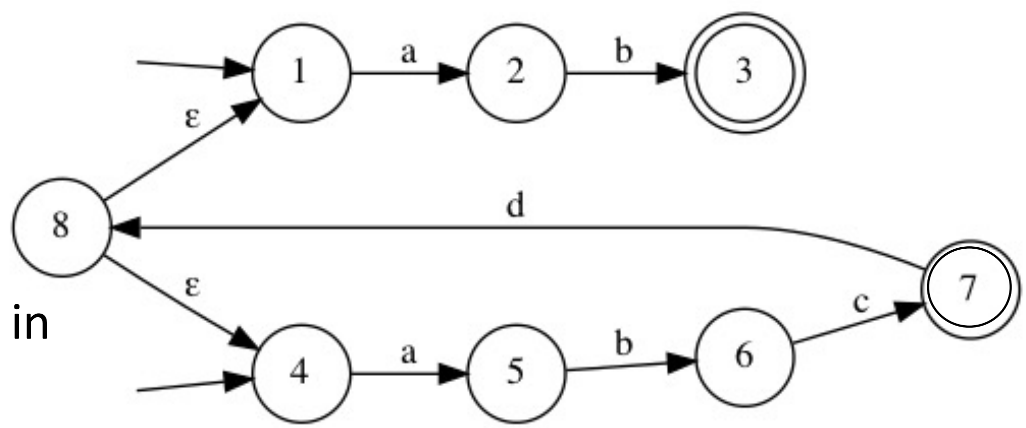
Sandiway Fong

Today's Topics

- FYI: there's an extra video on the state by-pass method on the course website
- Homework 11
- Beyond regular languages: $\{a^n b^n \mid n \geq 1\}$ and $\{1^n \mid n \text{ is prime}\}$
- A formal tool: the Pumping Lemma

Homework 11

- Consider the following NDFSA:
  - $\Sigma = \{a, b, c, d\}$ (alphabet)
  - $S = \{1, 4\}$ (start)
  - $F = \{3, 7\}$ (end)
  - $Q = \{1–8\}$ (states)
- Question 1:
  - what are the **five** shortest strings in L_{NDFSA}?



Homework 11

- Question 2:
  - $L_{NDFSA}{}^{R} = \{w^{R} \mid w \in L_{NDFSA}\}$
- **Example:**
  - (hypothetically) if ab is in L, ba is in L^{R}.
- Draw a FSA for $L_{NDFSA}{}^{R}$
- Make sure you label the start and final states properly.
- Your machine should be non-deterministic.
- Check your answer:
  - give the **five** shortest strings in $L_{NDFSA}{}^{R}$,
  - and compare with your answer in Q1.

Homework 11

- Question 3:
  - convert L_{NDFSA}^R to a DFSA
  - Using the *set-of-states* construction.
  - Show your sets (of states)
- Check your work:
  - the machines for questions 2 and 3 should accept the same language, but the DFSA should be deterministic and have no empty transitions (ε)!
  - How many states does the DFSA have?
  - How many start states?
  - How many end states?

Homework 11

- Question 4:
  - Consider the language $L_{NDFSA}{}^{RR} = \{w^R \mid w \in L_{NDFSA}{}^{R}\}$
  - Using *your* DFSA from Q3, construct the FSA for L^{RR}, taking care to label the start and final states properly.
- Note:
  - of course, $L_{NDFSA}{}^{RR}$ = the language L_{NDFSA} we started with.
- Point out where your resulting machine is non-deterministic.
- Compare your FSA with the machine we began with in Question 1.
  - Name two significant differences between the machines?

Homework 11

- Question 5:
  - Convert your machine constructed in Q4 to a DFSA.
  - Use the *set-of-states* construction again.
  - Show your sets (of states)
  - Compare your DFSA to the original machine from Q1:
    - how many states?
  - Do you think there could exist a machine for L_{NDFSA} (= $L_{NDFSA}{}^{RR}$) with fewer states than your new DFSA?
    - Explain your answer

Beyond Regular Languages

- Beyond regular languages
  - $a^n b^n$ = {ab, aabb, aaabbb, aaaabbbb, ... } $n \geq 1$
  - is not a regular language

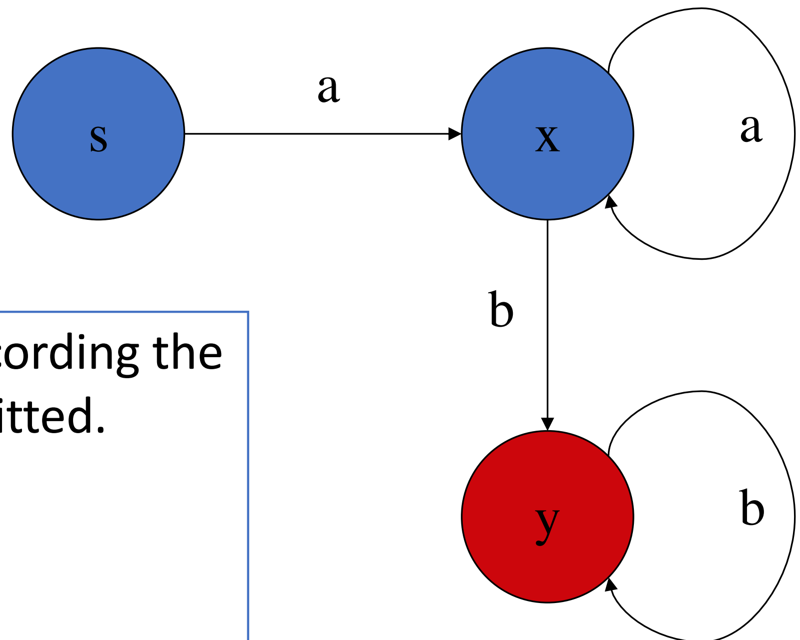
- That means no FSA, regex (or Regular Grammar) can be built for this set

- Informally, let's think about a FSA implementation …

1. We only have a finite number of states to play with …
2. We're only allowed simple free iteration (looping)

Beyond Regular Languages

- $L = a^+b^+$



Having a frequency table recording the number of visits is not permitted.
Not allowed:

```
%freq = ();

… $freq{$state}++;
```

A Formal Tool: The Pumping Lemma

[See also discussion in JM 16.2.1, pages 533–534]

- Let ***L*** be a regular language,
- then there exists a number $p > 0$
  - where p is a pumping length (*sometimes called a magic number*)

  such that every string w in ***L*** with $|w| \geq p$ can be written in the following form

  w = xyz
- with strings x, y and z such that $|xy| \leq p$, $|y| > 0$ and

  $xy^i z$ is in L
- for every integer $i \geq 0$.

BTW: there is also a pumping lemma for Context-Free Languages

A Formal Tool: The Pumping Lemma

Restated:

- For every (*sufficiently long*) string *w* in a regular language
- there is always a way to split the string into three adjacent sections, call them x, y and z, (y nonempty), i.e. w is x followed by y followed by z
- And y can be repeated as many times as we like (or omitted)
- And the modified string is still a member of the language

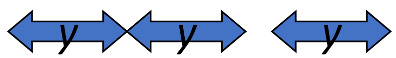
Essential Point!
To prove a language is non-regular: show that no matter how we split the string, there will be modified strings that can't be in the language.

A Formal Tool: The Pumping Lemma

- Example:
  - show that $a^n b^n$ is not regular
- Proof (by contradiction):
  - pick a sufficiently long string in the language
  - e.g. a..aab..bb (#a's = #b's)
  - Partition it according to ***w = xyz***
  - then show $xy^i z$ is **not** in L
  - i.e. *string does not pump*

A Formal Tool: The Pumping Lemma

aaaa..*aabbbb*..*bb*



Case 1: ***w = xyz, y straddles the ab boundary***
what happens when we pump y?

Case 2: ***w = xyz, y is wholly within the a's***
what happens when we pump y?

Case 3: ***w = xyz, y is wholly within the b's***
what happens when we pump y?

A Formal Tool: The Pumping Lemma

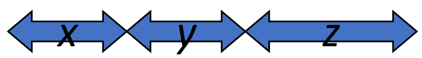
- Prime number testing

  prime number testing using Perl's extended "regular expressions"
  - Using unary notation, e.g. 5 = "11111"
  - /^(11+?)**\1**+$/ will match anything that's greater than 1 that's not prime

L = {1^n | n is prime} is not a regular language

A Formal Tool: The Pumping Lemma

1^n = 111..1111..11111 such that n is a prime number



For any split of the string
Pump y such that i = length(x+z), giving y^i

What is the length of string $w=xy^iz$ now?

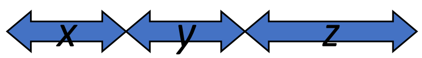
In x y^{xz} z , how many copies of xz do we have?
Answer is y+1
i.e. pumped number can be factorized into (1+|y|)|xz|

i.e., we can show any prime number can be pumped into a non-prime …

The resulting length is non-prime since it can be factorized

A Formal Tool: The Pumping Lemma

1^n = 111..1111..11111 such that *n* is a prime number



- Illustration of the calculation:

1111 1111 111 (eleven)

1111 1111 1111 1111 1111 1111 1111 1111 111

4 + 4*7 + 3

= 5*7

which isn't prime

- Another look:

1111 111 1111 (re-arrange eleven)

1111 111 1111 1111 1111 1111 111 111 111 111 (make 4 bundles of 4; 4 bundles of 3)

A Formal Tool: The Pumping Lemma

- Another angle to reduce the mystery, let's think in terms of FSA. We know:
  1. we can't control the loops
  2. we are restricted to a finite number of states
  3. assume (without loss of generality) there are no e-transitions

- Suppose there are a total of p states in the machine
- Supose we have a string in the language longer than p
- What can we conclude?

Answer: we must have visited some state(s) more than once!

Also: there must be a loop (or loops) in the machine!

Also: we can repeat or skip that loop and stay inside the language!