CSCB36

REVIEW SEMINAR!

Regular Languages review

- "Language" (L): a set of strings for a given alphabet
  - eg. {0, 101}, {1, 101, 10101, 1010101…}, $\{0^k1^{(k+2)}\}$, {all palindromes}
- "Regex" (R): a code specifying a set of strings (language)
  - eg. (0 + 101), 1+(01)*
  - Belongs to RE, the set of all regular expressions
- "Regular language": a language that can be encoded by a regex
  - Notated L(R)
  - eg. {1, 101, 10101, 1010101…} is regular
  - eg. {all palindromes} is **not** regular

REMEMBER: Regexes are STRINGS, languages are SETS

Regexes to languages

- For any two regexes S, T encoding languages L(S), L(T):
  - "S + T " encodes the language L(S) ∪ L(T) (union of sets)
  - "ST" encodes L(S)·L(T) (set concatenation)
    - Concatenation: all strings st where s ∈ L(S), t ∈ L(T)
  - "S*" encodes L(S)*
    - Kleene star: all combinations of elements of L(S) concatenated to each other in any amount. L(S)* always includes the empty string!
  - "Ø" encodes the empty set (no strings at all)
  - "ε" encodes {}, the set containing only the empty string
  - "a" encodes {a} for any single letter a in the alphabet
- Two regexes R, T are equivalent if the languages they encode are the same, i.e. L(R) = L(T)

Regex equivalences

2. State whether each of the following equivalences hold. If so, prove it formally. If not, disprove with a counterexample.

   (a) $RT \equiv (R + \epsilon)T$

   (b) $R^*T \equiv (R + \epsilon)^*T$

   (c) $RRT \equiv R(R + \epsilon)T$

Regex equivalences practice

2. State whether each of the following equivalences hold. If so, prove it formally. If not, disprove with a counterexample.

   (a) $RT \equiv (R + \epsilon)T$

   (b) $R^*T \equiv (R + \epsilon)^*T$

   (c) $RRT \equiv R(R + \epsilon)T$

Regex equivalences solution

(a) $RT \equiv (R + \epsilon)T$

Not equivalent. Let $R = 1, T = 0$. Then $0 \in L((R+\epsilon)T), 0 \notin L(RT)$.

(b) $R^*T \equiv (R + \epsilon)^*T$

Equivalent!

$$
\begin{aligned}
L(R^*T) &= L(R^*) \cdot L(T) && (14)\\
&= L(R)^* \cdot L(T) && (15)\\
&= (L(R) \cup \{\epsilon\})^* \cdot L(T) \text{ as } \{\epsilon\} \subset L(R)^* && (16)\\
&= L(R + \epsilon)^* \cdot L(T) && (17)\\
&= L((R^* + \epsilon)T) \text{ as wanted} && (18)
\end{aligned}
$$

(c) $RR^*T \equiv R^*T$

Not equivalent. Let $R = 1, T = 0$. Then $0 \in L(R^*T), 0 \notin L(RRT^*)$.

Structural induction review

The structure of a set defines the structure of our inductive proof.

Inductive set definition for S
- Base case: elements a, b, c are in S
- Inductive step: if x, y are in S, then z = f(x, y), w = g(x, y) are in S

Structural induction: prove P(x) holds for all x in S
- Base case: show P(a), P(b), P(c)
- Inductive step: for all x, y in S, show P(x), P(y) -> P(z), P(w)

Inductive set definition for RE
- Base case: regexes ∅, ε, single-letter a are in RE
- Inductive step: if regexes S, T are in RE, then regexes S + T, ST, S* are in RE

Structural induction on regexes: prove P(R) for all R in RE
- Base case: show P(∅), P(ε), P(a)
- Inductive step: for all S, T in RE, show P(S), P(T) -> P(S + T), P(ST), P(S*)

Structural Induction on Regular Languages

1. For some string $x \in \Sigma^*$, let $b(x)$ represent x in reverse. For example, if $x = CSCB36$, $b(x) = 63BSCS$.
   Let $BW(L)$ be the set of all backwards strings of L - for example, if $L = \{NESS, RAMEEN, BEN\}$, then $BW(L) = \{SSEN, NEEMAR, NEB\}$.
   Formally, $BW(L) = \{b(x) : x \in L\}$.
   Prove that if L is regular, then so is $BW(L)$.

Structural Induction on Regular Languages

Solution:
Define our predicate $B(R)$ as "there is a regex R' st $L(R') = BW(L(R))$".
We want to show that $B(R)$ holds for all $R \in RE$.

Base Cases:

(a) Let $R = \emptyset$, then $R' = \emptyset$.

(b) Let $R = \epsilon$, then $R' = \epsilon$.

(c) Let $R = a \in \Sigma^*$, then $R' = a$. (A single character is itself backwards).

Inductive Hypothesis:
Let $S, T \in RE$.
Suppose $B(S), B(T)$.
In other words, $\exists S', T'$ st $BW(L(S)) = L(S'), BW(L(T)) = L(T')$.

Inductive Step:
There are three cases: $R = S + T, R = ST, R = S^*$

Structural Induction on Regular Languages

(a) $R = S + T$. Let $R' = S' + T'$, then

$$
\begin{aligned}
BW(L(R)) &= BW(L(S+T)) \\
&= BW(L(S)) \cup BW(L(T)) \\
&\quad \text{as order does not matter for } + \\
&= L(S') \cup L(T') \\
&= L(S' + T') \\
&= L(R') \text{ as wanted}
\end{aligned}
$$

Structural Induction on Regular Languages

(b) $R = ST$. Let $R' = T'S'$, then

$$
\begin{aligned}
BW(L(R)) &= BW(L(ST)) && (1)\\
&= BW(L(T)) \cdot BW(L(S)) && (2)\\
&\text{as swapping the order of concatenation of the reverses, reverses the regex} && (3)\\
&= L(T') \cdot L(S') && (4)\\
&= L(T'S') && (5)\\
&= L(R') \text{ as wanted.} && (6)
\end{aligned}
$$

Structural Induction on Regular Languages

(c) $R = S^*$. Let $R' = S'^*$, then

$$
\begin{aligned}
BW(L(R)) &= BW(L(S^*)) && (7)\\
&= BW(L(S)^*) && (8)\\
&= BW(L(S))^* && (9)\\
&\text{as order does not matter for } * && (10)\\
&= L(S') && (11)\\
&= L(R')\text{as wanted} && (12)\\
& && (13)
\end{aligned}
$$

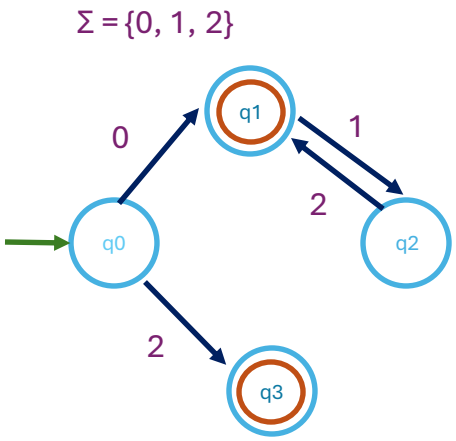
$\therefore$ by structural induction, $B(R)$ holds $\forall R \in RE$

$\therefore$ L is regular $\rightarrow$ $BW(L)$ is regular

DFSA Review

A DFSA M: 5-tuple (Q, Σ, δ, q0, F), with

- Q: the set of states
- Σ: the alphabet
- δ : Q × Σ → Q, the transition function
- q0 ∈ Q: the start state
- F ⊆ Q: the set of accepting states



- A string is accepted by M if it ends in an accepting state
- L(M): the language of all strings accepted by M

DFSA and regex practice

3. Let L be the set of strings where all occurrences of 00 are followed by an occurrence of 11 before the next 00 (including overlapping occurrences) or the end of the string.
   For example:
   Valid: $\epsilon, 0, 0011, 00101011, 0011001011$
   Invalid: $00, 1100, 0010011, 00011$

   Construct a regex and a DFSA for L.

DFSA and regex practice

(a) Regex: $(1^*0(1+01)^*11)^*$
Reasoning:

i. We can start with as many 1's (including none) as we like.

ii. If we encounter a 0:

A. If the next digit is a 1, no problem! We can keep going.

B. If the next digit is a 0, the digit after that MUST be a 1, as there can be no more occurrences of 00.

iii. Keep adding either 1's or 01s to the string until we get a 11.

iv. Repeat this entire pattern as long as we like.

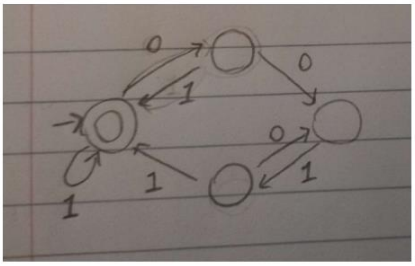
(b) We can apply similar logic to our construction of the DFSA. We will need:

i. A start / "all clear" state. (This will be our accepting state).

ii. A state that tracks if a first 0 has been read.

iii. A state that tracks if a second 0 has been read.

iv. Once in iii), "two" states that track if a first and a second 1 has been read. (The "second" state should be the initial accepting state).



NFSA Review

- Like DFSA but we can have…
  - Non-deterministic transitions
  - Epsilon transitions
- Remember the BIG RESULT
  - The following are equivalent for any language L
    - L is the language for some regular expressions
    - L is the language for some DFSA
    - L is the language for some NFSA
  - So NFSA's are not inherently more powerful

Preserving Regular Languages Review

- To show a function preserves regular languages
  - Given DFSA *M* that accepts L, describe how to construct FSA *M'* that accepts f(L)

- How many copies of M are needed in M'?
- What additional states (if any) are needed?
- What is the initial state of M'?
- What are the accepting states of M'?
- What transitions need to be added/removed/changed?
  - For each transition, make it clear what symbol (or epsilon) is being read.
  - For each transition, make it clear what state it goes from and what state it goes to.

NFSA Question

4. **[7 marks]** Let $\Sigma = \{0, 1\}$. Given a language L over Σ, we define $f(L)$ to be:

$$\{1y1 : 0y0 \in L\} \cup \{0y0 : 1y1 \in L\}$$

Let $M = (Q, \Sigma, \delta, s, F)$ be a DFSA.

Describe clearly how to construct an NFSA $M' = (Q', \Sigma, \delta', s', F')$ such that $\mathcal{L}(M') = f(\mathcal{L}(M))$.

No justification is required, but part marks may be given for it if your construction is incorrect.

NFSA Question – BAD SOLUTION (WHY?)

- Take 1 copy of M: M1
- *No additional states needed*
- The initial state of M' is the initial state of M1
- The accepting states of M' are the accepting states of M1
- For initial state in M1
  - Replacing outgoing 1-transitions with 0-transitions and vice versa
- For each accepting state in M1
  - Replacing outgoing 1-transitions with 0-transitions and vice versa

It doesn't enforce strings to have the same starting and ending character

NFSA Question – SOLUTION IDEA

- Take 2 copies of M: M1, M2
- Add two new states, qs and qe (start state, end state)
- The initial state of M' is qs
- The accepting state of M' is qe
- Add a 1-transition from qs to δ(s, 0) where s was the starting state in M1
- For each state q in M1 such that δ(q, 0) was originally an accepting state
  - Add a 1-transition from q to qe
- Add a 0-transition from qs to δ(s, 1) where s was the starting state in M2
- For each state q in M2 such that δ(q, 1) was originally an accepting state
  - Add a 0-transition from q to qe.

PUMPING LEMMA

If L is a REGULAR Language

There exists n>0 such that
for any string x ∈ L
where |x| >= *n*

there exists n > 0 such that
- (i) x = uvw for all x in L
- (ii) v not epsilon
- (iii) |uv| <= n
- (iv) uv^kw is in L for all k=0,1,2,3,4

By way of contradiction

Suppose L is regular, let n be as in the pumping lemma.

Let x = … ∈ L such that |x| >= n

Given (i), (ii), (iii) show that there exists k where uv^kw is not in L

PUMPING LEMMA QUESTION

If L is a REGULAR Language

There exists n>0 such that
for any string x ∈ L
where |x| >= n

4. **[7 marks]** Let $\Sigma = \{0, 1\}$. For arbitrary strings $x, y \in \Sigma^*$, we define $\#_y(x)$ to be $|\{(u, v) : x = uyv\}|$.
   Let $L_4 = \{x \in \Sigma^* : \#_{10}(x) \leq \#_{11}(x)\}$.
   Use the Pumping Lemma to prove that L_4 is **not** regular.

there exists n > 0 such that
- (i) x = uvw for all x in L
- (ii) v not epsilon
- (iii) |uv| <= n
- (iv) uv^kw is in L for all k=0,1,2,3,4

PUMPING QUESTION – SOLUTION IDEA

- Choose $x = 1^n(10)^n = uvw$
  - Show that x is in L, and $|x| >= n$
  - $\#_{11}(x)=n$, $\#_{10}(x)=n$ => $x \in L_4$
  - $|x| = 3n >= n$

PUMPING QUESTION – SOLUTION IDEA

- $|uv| <= n$
  - $u = 1^a$
  - $v = 1^b$
  - $w = 1^c(10)^n$
  - $b >= 1$
  - $a + b + c = n$

- Take $k = 0$
  - $uw = 1^a1^c(10)^n = 1^{(n-b)}(10)^n$
  - $\#_{11}(uw) = n-b-1+1 = n - b$
  - $\#_{10}(uw) = n$
  - $n-b < n$ (since $b>=1$)

- Thus *uw* not in L, language is not regular

PUMPING QUESTION – SOLUTION IDEA

- |uv| <= n
  - $u = 1^a$, $v = 1^b$, $w = 1^c(10)^n$
  - b >= 1, a + b + c = n
- Take k = 0
  - $uw = 1^a1^c(10)^n = 1^{(n-b)}(10)^n$
  - $\#_{11} = n-b-1+1$, $\#_{10} = n$
  - n-b < n (since b>=1)
  - Thus uw not in L, thus language is not regular

CFG Review

$$S \to ABC$$
$$A \to 11A, \epsilon$$
$$B \to 0B0, 0$$
$$C \to C10, C01, 1$$

A CFG G: 4-tuple (V, Σ, P, S), with

- V: A finite set of variables
- Σ: A finite set of terminals
- P: A finite set of productions (A -> b, A∈V, b∈(V∪Σ)*)
- S ∈ V: The start variable

- L(G): the language of all strings derived from S
- A language L' is said to be context-free (CFL) if L(G) = L' for some CFG G

CFG Review

A CFG $G = (V, \Sigma, P, S)$ is *right linear* if every production in P has one of the following forms:

1. $A \to \epsilon$
2. $A \to xB$

where $A, B \in V$ and $x \in \Sigma^*$.

G is *strict right-linear* if every production in P has one of following forms:

1. $A \to \epsilon$
2. $A \to xB$

where $A, B \in V$, $x \in \Sigma^*$, and $|x| \leq 1$.

CFG Review

- CFG are more powerful than Regular Expressions
- In fact, regular expressions are a subset of languages generated by CFG
  - THE BIG RESULT PLUS!
  - The following are equivalent
    - L = L(R) for some regular expression R
    - L = L(G) for some right-linear CFG G
    - L = L(G) for some strict right-linear CFG G

CFG Question

5. **[8 marks]** Let $\Sigma = \{3, 7\}$. For arbitrary strings $x, y \in \Sigma^*$, we define $\#_y(x)$ to be $|\{(u, v) : x = uyv\}|$.
Let $L_5 = \{x \in \Sigma^* : 2\#_{77}(x) = \#_{73}(x) + 1\}$. Let $L_0 = \{x \in \Sigma^* : 2\#_{77}(x) = \#_{73}(x)\}$.
Nick used the design below and the left-to-right method to create a CFG that generates L_5.
S generates L_5. For $i, j \in \Sigma$, A_{ij} generates $\{x \in L_0 : x$ starts with i, x ends with $j\}$.
Before anyone saw his CFG, the **Grammar Mangler** struck! Destroyed beyond recognition were all the S productions and all but one production for each of variables A_{33}, A_{73}, A_{77}. The A_{37} productions were untouched. Here is what remains of Nick's CFG.

$S \to \cdots$

$A_{33} \to 3A_{73}, \cdots$

$A_{37} \to 3A_{77}, 3A_{37}$
$A_{73} \to 7A_{77}A_{37}A_{33}, \cdots$

$A_{77} \to 7A_{77}A_{37}A_{37}, \cdots$

Please restore Nick's CFG where you see "$\cdots$". No justification is required.

CFG Question – SOLUTION

- $S \rightarrow 3S \mid 7A_{37}A_{77} \mid 7A_{77}A_{37} \mid 7A_{37}A_{73} \mid 7A_{77}A_{33}$
- $A_{33} \rightarrow 3A_{73} \mid 3A_{33} \mid 3$
- $A_{37} \rightarrow 3A_{77} \mid 3A_{37}$
- $A_{73} \rightarrow 7A_{77}A_{37}A_{33} \mid 7A_{37}A_{77}A_{33}$
- $A_{77} \rightarrow 7A_{77}A_{37}A_{37} \mid 7A_{37}A_{33}A_{77} \mid 7$