CSE 105
THEORY OF COMPUTATION

Friday, February 10, 2017

Winter 2017

http://cseweb.ucsd.edu/classes/wi17/cse105-ab/
Feedback Survey Results

- Pace: 83% say right speed
- Detail: 80% say just right
- Material: 84% say right level
- Exam Difficulty: 78% say about right
Feedback Survey Results

- Study Time: \( \frac{2}{3} \) of students spend 3 to 9 hours/week
- Homework: nearly everyone says helpful
- Homework frequency:
  - 87% say it prevents falling behind
  - 75% prefer it to longer weekly hw
Feedback Survey Results

Thanks for your suggestions for improvement. We will:

• post discussion worksheets online after Fridays
• provide full written solutions to discussion problems
• give a list of suggested practice problems from the textbook before each exam
• host a review session before the final exam
• drop your three lowest homework problem scores
Pushdown automata

- NFA + stack

At each step
1. **Transition** to new state based on current state, letter read from input, and top letter of stack.
2. May **push and/or pop** a letter on top of stack
Pushdown automata

- NFA + stack

Accept a string if there is some sequence of states and some sequence of stack contents which processes the entire input string and ends in an accepting state.
PDAs and CFGs are equivalently expressive

**Theorem 2.20**: A language is context-free if and only if some **nondeterministic** PDA recognizes it.

**Consequences**
- Quick proof that every regular language is context free
- To prove closure of class of CFLs under a given operation, can choose two modes of proof (via CFGs or PDAs) depending on which is easier
Example

\[ L = \{ a^i b^j c^k \mid i = j \text{ or } i = k, \text{ with } i,j,k \geq 0 \} \]

Which of the following strings are not in \( L \)?

A. b
B. abc
C. abbcc
D. aabcc
E. I don't know.
Example

\[ L = \{ a^i b^j c^k \mid i = j \text{ or } i = k, \text{ with } i, j, k \geq 0 \} \]

To design a CFG that generates \( L \)...
Example

\[ L = \{ a^i b^j c^k \mid i=j \text{ or } i=k, \text{ with } i,j,k \geq 0 \} \]

To design a CFG that generates \( L \)...

\[ L = \{ a^i b^j c^k \mid i=j, \text{ with } i,j,k \geq 0 \} \cup \{ a^i b^j c^k \mid i=k, \text{ with } i,j,k \geq 0 \} \]
Example

\[ L = \{ a^i b^j c^k \mid i=j \text{ or } i=k, \text{ with } i,j,k \geq 0 \} \]

To design a CFG that generates \( L \)…

\[ L = \{ a^i b^j c^k \mid i=j, \text{ with } i,j,k \geq 0 \} \cup \{ a^i b^j c^k \mid i=k, \text{ with } i,j,k \geq 0 \} \]

\[
S_1 \rightarrow S_1 c \mid T_1 \\
T_1 \rightarrow a T_1 b \mid \epsilon
\]
Example

\[ L = \{ a^i b^j c^k \mid i=j \text{ or } i=k, \text{ with } i,j,k \geq 0 \} \]

To design a CFG that generates \( L \)…

\[ L = \{ a^i b^j c^k \mid i=j, \text{ with } i,j,k \geq 0 \} \cup \{ a^i b^j c^k \mid i=k, \text{ with } i,j,k \geq 0 \} \]

\[
S_2 \rightarrow T_2 \mid aS_2c \\
T_2 \rightarrow T_2b \mid \varepsilon
\]
Example

\[ L = \{ a^i b^j c^k \mid i=j \text{ or } i=k, \text{ with } i,j,k \geq 0 \} \]

To design a CFG that generates \( L \)…

\[ L = \{ a^i b^j c^k \mid i=j, \text{ with } i,j,k \geq 0 \} \cup \{ a^i b^j c^k \mid i=k, \text{ with } i,j,k \geq 0 \} \]

\[ S \rightarrow S_1 \mid S_2 \]
Designing a PDA

L = \{ a^i b^j c^k \mid i = j \text{ or } i = k, \text{ with } i, j, k \geq 0 \}

*Informal description of PDA:*

How would you design an algorithm that, given a string, decides if it is in this set?
- What information do you need to track?
- How much memory do you need?
- Are you using non-determinism?
Designing a PDA

\[ L = \{ a^i b^j c^k \mid i=j \text{ or } i=k, \text{ with } i,j,k \geq 0 \} \]

Informal description of PDA:

- The PDA pushes a # to indicate the top of the stack, then starts reading a's, pushing each one onto the stack.
- The PDA guesses when it's reached the end of the a's and whether to match the number of a's to the number of b's or the number of c's.
- If trying to match number of b's with number of a's: PDA pops off a's for each b read. If there are more a's on the stack but no more b's being read, reject. When the end of the stack (#) is reached, we have read an equal number of a's and b's. If this is the end of the input or if any number of c's is read at this point, accept; otherwise, reject.
- If trying to match the number of c's with number of a's: first read any number of b's without changing stack contents and then nondeterministically guess when to start reading c's. For each c read, pop one a off the stack. When the end of the stack (#) is reached, we have read an equal number of a's and c's.
Designing a PDA

$L = \{ a^i b^j c^k | i=j \text{ or } i=k, \text{ with } i,j,k \geq 0 \}$

State diagram of PDA:
Conventions for PDAs

• Can "test for end of stack" without providing details
  • We can always push the end-of-stack symbol, $, at the start.

• Can "test for end of input" without providing details
  • Accept state only takes effect when entire input string has been read.

• Don't always need to provide a state transition diagram!
  • Too cumbersome. Give informal description instead. Still need to be precise enough that one could make the diagram from the description.
Other classes of languages?

Are all sets of strings over fixed finite alphabet $\Sigma$ context-free?

A. Yes, because the class of context-free languages is a strict superset of the regular languages.
B. Yes, because the set of all strings is regular.
C. Yes, because since $\Sigma$ is finite, all sets of strings over $\Sigma$ are regular.
D. No, because we can apply the Pumping Lemma.
E. No, because there are countably many context-free languages, but uncountably many languages.
Which specific language is not context-free?

A. \( \{ 0^n1^m0^n \mid m,n \geq 0 \} \)
B. \( \{ 0^n1^n0^n \mid n \geq 0 \} \)
C. \( \{ 0^n1^{2n} \mid n \geq 0 \} \)
D. \( \{ 0^n1^{2m} \mid m,n \geq 0 \} \)
E. I don't know.
Examples of non-context-free languages

• \{ a^n b^n c^n \mid 0 \leq n \}  \quad \text{Sipser Ex 2.36}
• \{ a^i b^j c^k \mid 0 \leq i \leq j \leq k \}  \quad \text{Sipser Ex 2.37}
• \{ w w \mid w \text{ is in } \{0,1\}^* \}  \quad \text{Sipser Ex 2.38}

To prove… Pumping lemma for CFLs (won't cover in CSE 105)
### Closure properties of ...

<table>
<thead>
<tr>
<th>The class of regular languages is closed under</th>
<th>The class of context-free languages is closed under</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Union</td>
<td>• Union</td>
</tr>
<tr>
<td>• Concatenation</td>
<td>• Concatenation</td>
</tr>
<tr>
<td>• Star</td>
<td>• Star</td>
</tr>
<tr>
<td>• Complementation</td>
<td>• Reversal</td>
</tr>
<tr>
<td>• Intersection</td>
<td>• Intersection</td>
</tr>
<tr>
<td>• Difference</td>
<td>• Complement</td>
</tr>
<tr>
<td>• Reversal</td>
<td>• Difference</td>
</tr>
</tbody>
</table>

**The class of CFLs is not closed under**

- Intersection
- Complement
- Difference
Context-free languages

Regular languages

Next model
Turing machines

- Unlimited input
- Unlimited (read/write) memory
- Unlimited time
Turing machine computation

- Read/write head starts at leftmost position on tape
- Input string written on leftmost squares of tape, rest is blank
- Computation proceeds according to transition function:
  - Given current state of machine, and current symbol being read,
    - transition to new state
    - write a symbol to its current position (overwriting existing symbol)
    - move the tape head L or R
- Computation ends if and when it enters either the accept or the reject state.
Language of a Turing machine

\[ L(M) = \{ w \mid \text{computation of } M \text{ on } w \text{ halts after entering the accept state} \} \]
i.e. \[ L(M) = \{ w \mid w \text{ is accepted by } M \} \]

Comparing TMs and PDAs, which of the following is true:
A. Both TMs and PDAs may accept a string before reading all of it.
B. A TM may only read symbols, whereas a PDA may write to its stack.
C. Both TMs and PDAs must read the string from left to right.
D. States in a PDA must be either accepting or rejecting, but in a TM may be neither.
E. I don't know.
Why is this model relevant?

• Division between program (CPU, state space) and data (memory) is a cornerstone of all modern computing.
• Unbounded memory is at outer limits of what modern computers (PCs, quantum computers, DNA computers) can implement.
• Simple enough to reason about, expressive enough to capture modern computation.
Formal definition of TM

A Turing machine is a 7-tuple \((Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})\) where \(Q, \Sigma, \Gamma\) are all finite sets and

1. \(Q\) is the set of states
2. \(\Sigma\) is the input alphabet (not containing blank symbol)
3. \(\Gamma\) is the tape alphabet (including blank symbol as well as all symbols in \(\Sigma\))
4. \(\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}\) is the transition function
5. \(q_0 \in Q\) is the start state
6. \(q_{\text{accept}} \in Q\) is the accept state
7. \(q_{\text{reject}} \in Q\) is the reject state

\(q_{\text{reject}} \neq q_{\text{accept}}\)
A Turing machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where $Q$, $\Sigma$, and $\Gamma$ are finite sets and

1. $Q$ is the set of states
2. $\Sigma$ is the input alphabet (including the blank symbol $\#$)
3. $\Gamma$ is the tape alphabet (including the blank symbol $\#$)
4. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function
5. $q_0 \in Q$ is the start state
6. $q_{accept} \in Q$ is the accept state
7. $q_{reject} \in Q$ is the reject state

What is the **input** of the transition function?

A. Current state and current character read
B. Current state and current character to write
C. Current state and next state
D. Current state and whether we came from L or R
E. I don't know.
Formal definition of TM

A Turing machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ where:

1. $Q$ is the set of states
2. $\Sigma$ is the input alphabet
3. $\Gamma$ is the tape alphabet (including blank symbol as well as all symbols in $\Sigma$)
4. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function
5. $q_0 \in Q$ is the start state
6. $q_{\text{accept}} \in Q$ is the accept state
7. $q_{\text{reject}} \in Q$ is the reject state

Turing machines are...

A. Deterministic
B. Nondeterministic
C. I don't know
Configurations of a TM

- Current state
- Current tape contents
- Current location of read/write head

\[ u \quad q \quad v \]

Current state is \( q \)
Current tape contents are \( uv \) (and then all blanks)
Current head location is first symbol of \( v \)
Configurations of a TM

- Current state
- Current tape contents
- Current location of read/write head

Start configuration on \( w \):
\[ q_0 \ w \]

Accepting configuration:
\[ u \ q_{\text{acc}} \ v \]

Rejecting configuration:
\[ u \ q_{\text{rej}} \ v \]

**Halting configuration**: any configuration that is either rejecting or accepting.

Current state is \( q \)

Current tape contents are \( uv \) (and then all blanks)

Current head location is first symbol of \( v \)
Transitioning between configurations

w is input, read/write head over the leftmost symbol of w

$q' = \delta(q, v_1)$

How does $uv$ compare to $u'v'$?
Language of a TM

\[ L(M) = \{ \text{w} | M \text{ accepts } \text{w}\} \]

= \{ w | \text{there is a sequence of configurations of M,} \\
\quad (C_1, \ldots, C_k), \text{ where} \\
\quad C_1 \text{ is start configuration of M on input w,} \\
\quad \text{each } C_i \text{ yields } C_{i+1}, \\
\quad \text{and } C_k \text{ is accepting configuration} \} \]

"The language of M"
"The language recognized by M"