Exam 2 — CSU 390 Theory of Computation — Fall 2007

Instructions

- The exam is open book: You may use your notes, homeworks, any handouts and solutions provided to you in this class, the text (Sipser), and any other paper-based references. You may not use any electronic devices such as laptops, cell phones, PDAs, etc.

- Please, write your answers in the blue books provided. Show all your work and indicate your final answers clearly.

Problem 1 [20 points]

Consider the following context-free grammar:

$$S \rightarrow xS | xStS |
\text{a}
$$

where \{a, t, x\} are terminals.

Prove that this grammar is ambiguous by providing the following:

- List all words of length less than 6 in the language \(L\) generated by this grammar.

  **Solution**

  (a) length 1: a
  (b) length 2: \(S \rightarrow xS \rightarrow xa\)
  (c) length 3: \(S \rightarrow xS \rightarrow xxS \rightarrow xxa\)
  (d) length 4: \(S \rightarrow xS \rightarrow xxS \rightarrow xxxS \rightarrow xxxa\)
  (e) length 4: \(S \rightarrow xStS \rightarrow xatS \rightarrow xata\)
  (f) length 5: \(S \rightarrow xS \rightarrow xxS \rightarrow xxxS \rightarrow xxxxS \rightarrow xxxxa\)
  (g) length 5: \(S \rightarrow xS \rightarrow xxStS \rightarrow xxatS \rightarrow xxata\)
  (h) length 5: \(S \rightarrow xStS \rightarrow xxStS \rightarrow xxatS \rightarrow xxata\)
  (i) length 5: \(S \rightarrow xStS \rightarrow xStxS \rightarrow xatxS \rightarrow xatxa\)

- Show a string \(w\) in the language \(L\) that has (at least) two leftmost derivations in this language.

  **Solution**

  String \(xxata\) has two leftmost derivations shown in (g) and (h).
• Show the two derivations for the string \( w \).

**Solution**

(g) length 5: \( S \rightarrow xS \rightarrow xS \rightarrow xxatS \rightarrow xxata \)

(h) length 5: \( S \rightarrow xS \rightarrow xxS \rightarrow xxatS \rightarrow xxata \)

• Show the parse trees corresponding to these derivations.

```
S    S
/    /
/    /
/    /
x    x
S    S
/    /
/    /
/    /
x    x
S    S
/    /
/    /
/    /
x    x
S    S
/    a
/    a
a    a
```

• Give a concise description of the language generated by this grammar.

**Solution**

Each string starts with \( x \). One option is a finite number of \( x \)s followed by \( a \), by using the rule \( S \rightarrow xS \) exclusively. The rule \( S \rightarrow xS \) introduces ambiguity, because we can first apply the rule \( S \rightarrow xS \) then apply the rule \( S \rightarrow xS \) to the first \( S \), or start with \( S \rightarrow xS \) and apply the rule \( S \rightarrow xS \) to the first \( S \). This ambiguity persists for an arbitrary number of levels in the parse tree.

**Problem 2 [20 points]**

Give context-free grammars that generate the following languages over \( \{a, b\} \). In each case annotate the rules to indicate what each rule generates.

- \( L_1 = \{a^n b^{n+2} | n \geq 0\} \)

**Solution**

\( S \rightarrow aSb|bb \)

The first rule generated an equal number of \( a \)s and \( b \)s, actually arranged as \( a^n b^n \), with the variable \( S \) remaining in the middle. 

The
second rule then replaces $S$ by $bb$, thus adding two more $b$ to the previously balanced string.

- $L_2 = \{w||w| \text{ is odd and the middle symbol is } a\}$

Solution

$$S \rightarrow aSa| aSb | bSa | bSb | a$$

The first four rules add one terminal on each side of the middle, covering the four possible combinations of the two letters $a$ and $b$. The fifth rule converts the variable in the middle to the terminal $a$, assuring both of the requirements — that the string length is odd, and that it is $a$.

Problem 3 [15 points]

Give a state transition diagram and a brief informal description for a pushdown automaton that recognizes the following language:

$L_3 = \{ a^m b^{m+n} c^n | m, n \geq 0 \}$

Solution

- Start by pushing $\$ \$ onto the stack to mark the bottom of the stack and move to the state 1.
- In the state 1 push all $a$s onto the stack.
- When the first $b$ is about to be read move to the state 2.
- Pop $a$s and match them with input $b$s until the bottom of the stack is reached.
• Pop and push back the bottom of the stack marker $ and move to the state 3.

• Push the remaining bs in the input onto the stack.

• When the first c is about to be read move to the state 4.

• Pop bs and match them with input cs until the bottom of the stack is reached.

• Pop the the bottom of the stack marker $ and move to the Accept state.

• Add the transition that rejects any additional symbol after the bottom of the stack has been reached in the state 4.

**Alternate Solution**
Start with the CFG that recognizes the language $L_3$:

\[
S \rightarrow TU
\]

\[
T \rightarrow aTb|\epsilon
\]

\[
U \rightarrow bUc|\epsilon
\]

and construct the PDA following the technique described in the book on page 118:
Problem 4 [15 points]

Prove that the language \( L_4 = \{a^n b^i c^k \mid k > n, k > i \} \) is not context-free.

Solution

Suppose the pumping length is \( p \). Choose the following string \( w = a^p b^p c^{p+1} \).

Then any substring \( s = vxy \) of \( w \) of length \( |s| \leq p \) must have one of the following forms:

- \( s = a^m \) where \( m \leq p \). Then pumping up would produce a string \( a^{m'} b^p c^{p+1} \) with \( m' > (p+1) \) and so the pumped up string would not be in \( L_4 \).

- \( s = b^m \) where \( m \leq p \). Then pumping up would produce a string \( a^p b^{m'} c^{p+1} \) with \( m' > (p+1) \) and so the pumped up string would not be in \( L_4 \).

- \( s = c^m \) where \( m \leq p \). This cannot be pumped down, as in the resulting string \( a^p b^p c^{m'} \) we have \( m' \leq p \) and so the pumped down string would not be in \( L_4 \).

- \( s = a^{m_1} b^{m_2} \). This string cannot be pumped up, as it would either produce \( a \) and \( b \)s out of order, or produce a string with more \( a \)s and \( b \)s than \( c \)s and so the pumped up string would not be in \( L_4 \).

- \( s = b^{m_1} c^{m_2} \). We can divide \( s \) in one of the following ways:
  - \( s = vxy = b^{m_1'} \circ x \circ c^{m_2'} \), but then the string cannot be pumped down, as there would no longer be more \( c \)s than \( a \)a.
  - \( |y| = 0 \) and \( v = b^{m_1} \), or \( y = b^{m_1} \) and \( |v| = 0 \) — but then the string cannot be pumped up, as there would be more \( b \)s than \( c \)s.
  - \( |y| = 0 \) and \( v = c^{m_2} \), or \( y = c^{m_2} \) and \( |v| = 0 \) — but then the string cannot be pumped down, as there would no longer be more \( c \)s than \( b \)s or \( a \)s.
  - Finally, if either \( v = b^{m_1} c^{m_2} \) or \( y = b^{m_1} c^{m_2} \) then pumping up the string would produce \( b \)s and \( c \)s out of order.
Problem 5 [15 points]

- Give a state transition diagram for a TM $M$ with input alphabet $\{a, b\}$ that accepts all strings starting with $a$, rejects the empty string, and loops on all other strings. Your diagram may use the implicit-reject convention.

Solution

![Diagram](image)

- Is $L(M)$, the language recognized by your TM $M$, a decidable language? Prove your answer.

Solution

This is a decidable language. We can build a Turing machine that would just eliminate the transition to the state 3 and replace it with an implicit rejection.
Problem 6 [15 points]

- Construct a TM to test for equality of two strings over the alphabet \{a, b\}, where the strings are separated by a cell containing \#.

Solution
• Describe in English the actions of your TM.

- In state 0 read a character and write \( x \). Follow to the state 1 if the input was \( b \) and to state 5 if the input was \( a \).
- From state 1 keep reading all the letters of the first string without writing anything new and moving to the right each time.
- When \#\ appears on the input, move right and transition to state 2.
- Keep reading over the input that has been marked with \( x \)s.
- The first letter after \( x \)s should match the \( b \) that we have read in the first string. If it does, write \( x \) on the tape and move to the left, and transition to the state 3.
- Keep reading over all marked letters of the second string, until you find \#.
- Read \# and transition to the state 4.
- Read over the letters of the first string moving left until you find the rightmost marked one.
- When you find \( x \) move to the right to look at the first un-matched letter of the first string and transition back to state 0.
- For matching letter \( a \) we follow the same process through states 5, 6, 7, 8 and back to 0.
- If all the letters in the first string have been matched (or the first string was empty) read the \# in the state 0 and transition to the state 9.
- In the state 9 read all the marked letters of the second string, moving to the right.
- If the letter after all marked ones is a blank, then all the letters in the second string have been matched with corresponding letters in the first string and we move to the state 10, the \( ACCEPT \) state.
- In all states, encountering a letter not explicitly expected leads to the \( REJECT \) state.