Ambiguous Grammars and Compactification

Mridul Aanjaneya

Stanford University

July 17, 2012

Midterm Review

- Mathematical Induction and Pigeonhole Principle
- Finite Automata (DFA's, NFA's, ε-NFA's)
- Equivalence of DFA's, NFA's and ε-NFA's
- Regular Languages and Closure Properties
  - Union, Concatenation, Kleene Closure, Intersection, Difference, Complement, Reversal, (Inverse) Homomorphisms
- Regular Expressions and Equivalence with Automata
- Pumping Lemma for DFA's
- Efficient State Minimization for DFA's
- Context-Free Languages and Parse Trees

Homework #1

Problem 1

Given 8 distinct natural numbers, none greater than 15, show that at least three pairs of them have the same positive difference.

- The pairs need not be disjoint as sets.

- 8 distinct natural numbers $\Rightarrow$ 28 pairs.
- 14 possible differences.
- The difference 14 can only be achieved by 1 pair $\Rightarrow$ PHP!

- **Common Mistakes:**
  1. Not mentioning PHP.

Homework #1

Problem 2

Prove that the number $111\ldots11$ (243 ones) is divisible by 243.

- Let α_n denote the number $111\ldots11$ (3^n ones).
- Proof by induction.

$$\alpha_{n+1} = \alpha_n 10^{2\cdot 3^n} + \alpha_n 10^{3^n} + \alpha_n$$

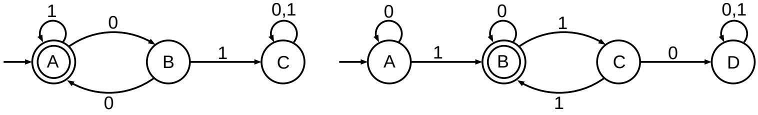
- $10^{2\cdot 3^n}$ and 10^{3^n} leave remainder 1 modulo 3.

$$\Rightarrow \alpha_{n+1} \equiv 3 \cdot \alpha_n (\text{mod } 3) \equiv 0 (\text{mod } 3^{n+1})$$

- **Common Mistakes:**
  1. Using divisibility of sum of digits modulo 3 without proof.
  2. Not stating general claim implies $n = 5$.

Problem 3

Show that the language L consisting of runs of even numbers of 0's and runs of odd numbers of 1's is regular.

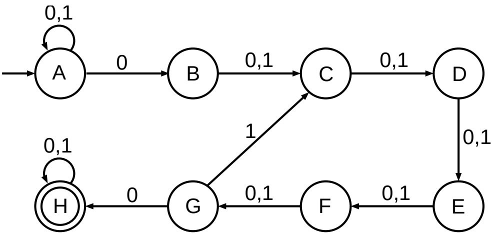


- **Common Mistakes:**
  1. Not having a reject state that loops on all inputs.
  2. Inconsistent use of empty string.

Homework #1

Problem 4

Construct an NFA for the language L over $\{0,1\}$ such that each string has two 0's separated by a number of positions that is a non-zero multiple of 5.

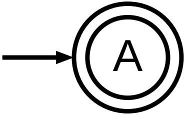


- **Common Mistakes:**
  1. Looping back incorrectly.
  2. Not enforcing that the first and last transitions are zeroes.

Homework #1

Problem 5

Define an NFA N_2 from a given NFA N_1 by reversing the final/non-final states. Is $L(N_2)$ the complement of $L(N_1)$?



- **Common Mistakes:**
  1. Overcomplicated thoughts.
  2. Proving its true!

Recap: Context-Free Grammars

- **Terminals:** symbols of the alphabet of the language being defined.
- **Variables (nonterminals):** a finite set of other symbols, each of which represents a language.
- **Start symbol:** the variable whose language is the one being defined.

Example: Formal CFG

- Here is a formal CFG for $\{0^n1^n \mid n \geq 1\}$.
- Terminals $= \{0,1\}$.
- Variables $= \{S\}$.
- Start symbol $= S$.
- Productions $=$
  $S \rightarrow 01$
  $S \rightarrow 0S1$

Recap: Leftmost Derivations

- Say $wA\alpha \Rightarrow_{lm} w\beta\alpha$ if w is a string of terminals only and $A \rightarrow \beta$ is a production.
- Also, $\alpha \Rightarrow^*_{lm} \beta$ if α becomes β by a sequence of zero or more $\Rightarrow_{lm}$ steps.
- Balanced parantheses grammar:

$$S \rightarrow SS \mid (S) \mid ()$$

- $S \Rightarrow_{lm} SS \Rightarrow_{lm} (S)S \Rightarrow_{lm} (())S \Rightarrow_{lm} (())()$
- Thus, $S \Rightarrow^*_{lm} (())()$

Recap: Rightmost Derivations

- Say $\alpha Aw \Rightarrow_{rm} \alpha\beta w$ if w is a string of terminals only and $A \to \beta$ is a production.
- Also, $\alpha \Rightarrow^*_{rm} \beta$ if α becomes β by a sequence of zero or more $\Rightarrow_{rm}$ steps.
- Balanced parantheses grammar:

$$S \to SS \mid (S) \mid ()$$

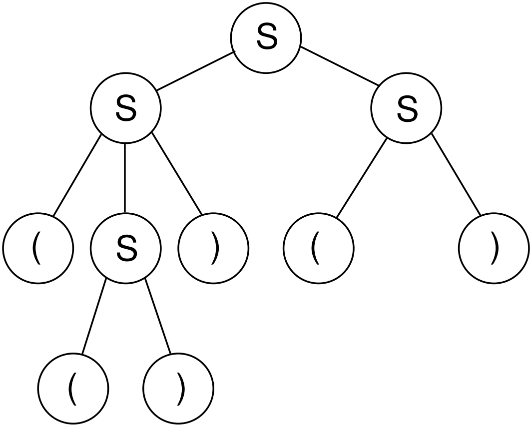
- $S \Rightarrow_{rm} SS \Rightarrow_{rm} S() \Rightarrow_{rm} (S)() \Rightarrow_{rm} (())()$
- Thus, $S \Rightarrow^*_{rm} (())()$

Recap: Parse Trees

- Parse trees are trees labeled by symbols of a particular CFG.
- **Leaves:** labeled by a terminal or ε.
- **Interior nodes:** labeled by a variable.
  - Children are labeled by the right side of a production for the parent.
- **Root:** must be labeled by the start symbol.

Example: Parse Tree

$$S \rightarrow SS \mid (S) \mid ()$$



Recap: Ambiguous Grammars

- A CFG is ambiguous is there is a string in the language that is the yield of two or more parse trees.
- **Example:** $S \to SS \mid (S) \mid ()$
- Two parse trees for ()()()!

Ambiguous Grammars

- Ambiguity is a property of grammars not languages.
- For the balanced parentheses language, here is another CFG which is unambiguous:

$$\mathsf{B} \rightarrow (\mathsf{RB} \mid \varepsilon$$
$$\mathsf{R} \rightarrow \,) \mid (\mathsf{RR}$$

- **Note:** R generates strings that have one more right parentheses than left.

Unambiguous Grammars

$$B \to (RB \mid \varepsilon \quad R \to \,) \mid (RR$$

- Construct a unique leftmost derivation for a given balanced string of parentheses by scanning the string from left to right.
- If we need to expand B, then use $B \to (RB$ if the next symbol is (and ε if at the end.
- If we need to expand R, use $R \to \,)$ if the next symbol is) and (RR if it is (.

The Parsing Process

- $(())()$

$$
\begin{aligned}
B &\rightarrow (RB \rightarrow ((RRB \rightarrow (()RB \rightarrow (())B \rightarrow (())(RB \\
&\rightarrow (())()B \rightarrow (())()
\end{aligned}
$$

LL(1) Grammars

- As an aside, a grammar such as

$$B \rightarrow (RB \mid \varepsilon$$
$$R \rightarrow) \mid (RR$$

  where you can always figure out the production to use in a leftmost derivation by scanning the given string left-to-right and looking only at the next one symbol is called LL(1).
    - Leftmost derivation, left-to-right scan, one symbol of lookahead.
- Most programming languages have LL(1) grammars.
- LL(1) grammars are never ambiguous.

Inherent Ambiguity

- It would be nice if for every ambiguous grammar, there was some way to fix the ambiguity, as we did for the balanced parentheses grammar.
- Unfortunately, certain CFL's are inherently ambiguous, meaning that every grammar for the language is ambiguous.

Example: Inherent Ambiguity

- The language $L = \{0^i1^j2^k \mid i = j \text{ or } j = k\}$
- Intuitively, at least some of the strings of the form $0^n1^n2^n$ must be generated by two different parse trees, one based on checking the 0's and 1's, the other based on checking the 1's and 2's.

One Possible Ambiguous Grammar

$$
\begin{aligned}
S &\rightarrow AB|CD \\
A &\rightarrow 0A1|01 \\
B &\rightarrow 2B|2 \\
C &\rightarrow 0C|0 \\
D &\rightarrow 1D2|12
\end{aligned}
$$

- A generates equal numbers 0's and 1's.
- B generates any number of 2's.
- C generates any number of 0's.
- D generates equal numbers 1's and 2's.

One Possible Ambiguous Grammar

$$
\begin{aligned}
S &\rightarrow AB|CD \\
A &\rightarrow 0A1|01 \\
B &\rightarrow 2B|2 \\
C &\rightarrow 0C|0 \\
D &\rightarrow 1D2|12
\end{aligned}
$$

- And there are two derivations of every string with equal numbers of 0's, 1's and 2's. e.g.:

$$
\begin{aligned}
S &\rightarrow AB \rightarrow 01B \rightarrow 012 \\
S &\rightarrow CD \rightarrow 0D \rightarrow 012
\end{aligned}
$$

Example #1

Problem

Show that the language of all palindromes over $\{0,1\}$ is context-free.

Example #1

Problem

Show that the language of all palindromes over $\{0,1\}$ is context-free.

$$P \to \varepsilon \mid 0 \mid 1 \mid 0P0 \mid 1P1$$

Example #2

Problem

Give a context-free language for the regular expression $0^*1(0+1)^*$.

Example #2

Problem

Give a context-free language for the regular expression $0^*1(0+1)^*$.

$$\begin{aligned} S &\rightarrow A1B \\ A &\rightarrow 0A|\varepsilon \\ B &\rightarrow 0B|1B|\varepsilon \end{aligned}$$

Variables that derive nothing

- Consider: $S \rightarrow AB$, $A \rightarrow aA$, $B \rightarrow AB$
- Although A derives all strings of a's, B derives no terminal strings.
- Thus, S derives nothing, and the language is empty!

Testing if a variable derives a terminal string

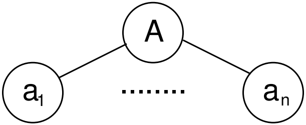
- **Basis:** If there is a production $A \rightarrow w$, where w has no variables, then A derives a terminal string.
- **Induction:** If there is a production $A \rightarrow \alpha$, where α consists only of terminals and variables known to derive a terminal string, then A derives a terminal string.

Testing

- Eventually, we can find no more variables.
- An easy induction on the order in which variables are discovered shows that each one truly derives a terminal string.
- Conversely, any variable that derives a terminal string will be discovered by this algorithm.

Proof of Converse

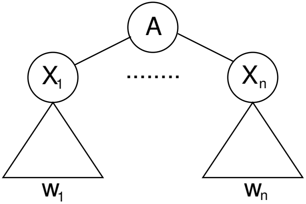
- The proof is an induction on the height of the least-height parse tree by which a variable A derives a terminal string.
- **Basis:** Height = 1. Tree looks like:



- Then the basis of the algorithm tells us that A will be discovered.

Induction for Converse

- Assume **IH** for parse trees of height $< h$, and suppose A derives a terminal string via a parse tree of height h:



- By **IH**, those X_i's that are variables are discovered.
- Thus, A will also be discovered, because it has a right side of terminals and/or discovered variables.

Algorithm to Eliminate Variables that Derive Nothing

1. Discover all variables that derive terminal strings.
2. For all other variables, remove all productions in which they appear, either on the left or on the right.

Example: Eliminate Variables

$$S \rightarrow AB \mid C,\ A \rightarrow aA \mid a,\ B \rightarrow bB,\ C \rightarrow c$$

- **Basis:** A and c are identified because of $A \rightarrow a$ and $C \rightarrow c$.
- **Induction:** S is identified because of $S \rightarrow C$.
- Nothing else can be identified.
- **Result:**

$$S \rightarrow C,\ A \rightarrow aA \mid a,\ C \rightarrow c$$

Unreachable Symbols

- Another way a terminal or a variable deserves to be eliminated is if it cannot appear in any derivation from the start symbol.
- **Basis:** We can reach S (the start symbol).
- **Induction:** If we can reach A, and there is a production

$$\mathsf{A} \to \alpha,$$

then we can reach all symbols of α.

Unreachable Symbols

- Easy inductions in both directions show that when we can discover no more symbols, then we have all and only the symbols that appear in derivations from S.
  - Left as exercises.
- **Algorithm:** Remove from the grammar all symbols not discovered reachable from S and all productions that involve these symbols.

Eliminating Useless Symbols

- A symbol is useful if it appears in some derivation of some terminal string from the start symbol.
- Otherwise it is useless. Eliminate all useless symbols by:
  1. Eliminating symbols that derive no terminal string.
  2. Eliminating unreachable symbols.

Useless Symbols

$$S \rightarrow AB, A \rightarrow C, C \rightarrow c, B \rightarrow bB$$

- If we eliminated unreachable symbols first, we would find everything is reachable.
- A, C and c would never get eliminated.

Why It Works

- After step (1), every symbol remaining derives some terminal string.
- After step (2), the only symbols remaining are all derivable from S.
- In addition, they still derive a terminal string, because such a derivation can only involve symbols reachable from S.

Epsilon Productions

- We can almost avoid using productions of the form $A \rightarrow \varepsilon$ (called ε-productions).
    - The problem is that ε cannot be in the language of any grammar that has no ε-productions.

Theorem

If L is a CFL, then $L - \{\varepsilon\}$ has a CFG with no ε-productions.

Nullable Symbols

- To eliminate ε-productions, we first need to discover the nullable symbols = variables A such that $A \Rightarrow^* \varepsilon$.
- **Basis:** If there is a production $A \rightarrow \varepsilon$, then A is nullable.
- **Induction:** If there is a production $A \rightarrow \alpha$, and all symbols of α are nullable, then A is nullable.

Example: Nullable Symbols

$$S \to AB,\ A \to aA \mid \varepsilon,\ B \to bB \mid A$$

- **Basis:** A is nullable because of $A \to \varepsilon$.
- **Induction:** B is nullable because of $B \to A$.
- Then, S is nullable because of $S \to AB$.

Proof of Algorithm: Nullable Symbols

- Proof is very much like that for the algorithm for testing variables that derive terminal strings.
- Left to the imagination!

Eliminating ε-productions

- **Key idea:** turn each production

$$A \rightarrow X_1 \ldots X_n$$

  into a family of productions.
- For each subset of nullable X's, there is one production with those eliminated from the right side in advance.
  - Except, if all X's are nullable, do not make a production with ε as the right hand side.

Example: Eliminating ε-productions

$$S \to ABC,\ A \to aA \mid \varepsilon,\ B \to bB \mid \varepsilon,\ C \to \varepsilon$$

- A, B, C and S are all nullable.
- New grammar:

$$\begin{aligned} S &\to \cancel{ABC} \mid AB \mid \cancel{AC} \mid \cancel{BC} \mid A \mid B \mid \cancel{C} \\ A &\to aA \mid a \\ B &\to bB \mid b \end{aligned}$$

- **Note:** C is now useless, eliminate its productions.