Normal Forms and Parsing

CS154
Chris Pollett
Outline

• Chomsky Normal Form
• The CYK Algorithm
• Greibach Normal Form
Chomsky Normal Form

- To get an efficient parsing algorithm for general CFGs it is convenient to have them in some kind of normal form.
- Chomsky Normal Form is often used.
- A CFG is in **Chomsky Normal Form** if every rule is of the form $A \rightarrow BC$ or of the form $A \rightarrow a$, where $A, B, C$ are any variables and $a$ is a terminal. In addition the rule $S \rightarrow \lambda$ is permitted.
Conversion to Chomsky Normal Form

Any CFL L can be generated by a CFG in Chomsky Normal Form

Proof Let G be a CFG for L. First we add a new start variable and rule $S_0 \rightarrow S$. This guarantees the start variable does not occur on the RHS of any rule. Second we remove any $\lambda$ -rules $A \rightarrow \lambda$ where A is not the start variable. Then for each occurrence of A on the RHS of a rule, say $R \rightarrow uAv$, we add a rule $R \rightarrow uv$. We do this for each occurrence of an A. So for $R \rightarrow uAvA\lambda$, we would add the rules $R \rightarrow uvA\lambda$, $R \rightarrow uAv\lambda$, $R \rightarrow uv\lambda$. If we had the rule $R \rightarrow A$, add the rule $R \rightarrow \lambda$ unless we previously removed the rule $R \rightarrow \lambda$. Then we repeat the process with R. Next we handle unit rule $A \rightarrow B$. To do this, we delete this rule and then for each rule of the form $B \rightarrow u$, we add then rule $A \rightarrow u$, unless this is a unit rule that was previously removed. We repeat until we eliminate unit rules. Finally, we convert all the remaining rules to the proper form. For any rule $A \rightarrow u_1u_2 \ldots u_k$ where $k \geq 3$ and each $u_i$ is a variable or a terminal symbol, we replace the rule with $A \rightarrow u_1A_1$, $A_1 \rightarrow u_2A_2$, $\ldots$ $A_{k-2} \rightarrow u_{k-1}u_k$. For any rule with $k=2$, we replace any terminal with a new variable $U_i$ and a rule $U_i \rightarrow u_i$. 
Example

- Use the algorithm to convert: $S \rightarrow Aba$, $A \rightarrow aab$, $B \rightarrow Ac$ to Chomsky Normal Form
  - Step 1: Add new start variable to get:
    - $S_0 \rightarrow S$, $S \rightarrow Aba$, $A \rightarrow aab$, $B \rightarrow Ac$
  - Step 2: Remove $\lambda$ rules. In this case, there aren’t any so we still have:
    - $S_0 \rightarrow S$, $S \rightarrow Aba$, $A \rightarrow aab$, $B \rightarrow Ac$
  - Step 3: Remove unit rules. Only have one, involving $S_0$.
    - $S_0 \rightarrow Aba$, $S \rightarrow Aba$, $A \rightarrow aab$, $B \rightarrow Ac$
  - Step 4: Split up rules with RHS of length longer than 2:
    - $S_0 \rightarrow AC_1$, $C_1 \rightarrow ba$, $S \rightarrow AD_1$, $D_1 \rightarrow ba$, $A \rightarrow aE_1$, $E_1 \rightarrow ab$, $B \rightarrow Ac$
  - Step 5: Put each rule with RHS of length 2 into the correct format:
    - $S_0 \rightarrow AC_1$,
    - $C_1 \rightarrow B_1 A_1$, $B_1 \rightarrow b$, $A_1 \rightarrow a$,
    - $S \rightarrow AD_1$,
    - $D_1 \rightarrow B_2 A_2$, $B_2 \rightarrow b$, $A_2 \rightarrow a$,
    - $A \rightarrow A_3 E_1$, $A_3 \rightarrow a$
    - $E_1 \rightarrow A_4 B_4$, $B_4 \rightarrow b$, $A_4 \rightarrow a$,
    - $B \rightarrow AC_2$, $C_2 \rightarrow c$

The answer
Introduction to Cocke-Younger-Kasami (CYK) algorithm (1960)

- This is an $O(n^3)$ algorithm to check if a string $w$ is can be generated by a CFG in Chomsky Normal Form.
- As cubic algorithms tend to be slow, in practice people use algorithms based on restricted types of CFGs with a fixed amount of lookahead. Either top down LL parsing or bottom-up LR parsing. These algorithms are based on the PDA model.
- There have been improvements to CYK algorithm which reduce the run-time slightly below cubic ($n^{2.8}$) and to quadratic in the case of an unambiguous grammar.
The CYK algorithm

- The idea is to build a table such that table(i,j) contains those variables that can generate the substring of w start at location i until location j.

Algorithm:
On input w= $w_1w_2...w_n$:
1. If $w = \varepsilon$ and S--> $\varepsilon$ is a rule accept.
2. For $i = 1$ to $n$: [set up the substring of length 1 case]
3. For each variable $A$:
4. Test whether $A$--> $b$ is a rule, where $b=w_i$
5. If so, place $A$ in table(i,i).
6. For $l = 2$ to $n$: [Here $l$ is a length of a substring]
7. For $i = 1$ to $n - l + 1$: [i is the start of the substring]
8. Let $j = i + l - 1$, [j is the end of the substring]
9. For $k = i$ to $j-1$: [k is a place to split substring]
10. For each rule $A$-->BC
11. If table(i,k) contains B and table(k+1, j) contains C put A in table(i,j).
12. If S is in table(1,n) accept. Otherwise, reject.
Example

- Consider the context free grammar $S \rightarrow AT$, $S \rightarrow c$, $T \rightarrow SB$, $A \rightarrow a$, $B \rightarrow b$.
- Let’s look at the steps CYK would do to check if aacbb was in the language.
- We’ll abbreviate table$(i,j)$ as $T(i,j)$
- First, lines 2-5 would be used to set $T(1,1) = \{ A \}$, $T(2,2) = \{ A \}$, $T(3,3) = \{ S \}$, $T(4,4) = \{ B \}$, $T(5,5) = \{ B \}$.
- The $l=2$ pass of lines 8-11 then fills in the table for substrings of length 2. We get $T(1,2)=\{ \}$, $T(2,3)=\{ \}$, $T(3,4) = \{ T \}$, $T(4,5) = \{ \}$. Notice we added $T$ to $T(3,4)$ because $T(3,3)$ was $\{ S \}$ and $T(4,4)$ was $\{ B \}$ and we have a rule $T \rightarrow SB$.
- The $l=3$ pass of lines 8-11 then fills the table for substrings of length 3. We get $T(1,3)=\{ \}$, $T(2,4)=\{ S \}$, $T(3,5)=\{ \}$. Here $T(2,4)=\{ S \}$, since $T(2,2)=\{ A \}$ and $T(3,4)=\{ T \}$ and we have the rule $S \rightarrow AT$.
- The $l=4$ pass of lines 8-11 then fills the table for substrings of length 4. We get $T(1,4)=\{ \}$ and $T(2,5)=\{ T \}$. This follows as $T(2,4)=\{ S \}$ and $T(5,5)=\{ B \}$ and $T \rightarrow SB$.
- Finally, when the $l=5$ pass of lines 8-11 then fills the table for substrings of length 5. i.e., the whole string aacbb. In this case, $T(1,5)=\{ S \}$ since $T(1,1)=\{ A \}$ and $T(2,5)=\{ T \}$ and we have the rule $S \rightarrow AT$. As $T(1,5)$ has the start variable we know the string aacbb is generated by the whole grammar.
Greibach Normal Form

- A CFG is said to be in Greibach Normal Form if all productions are of the form $A \rightarrow ax$ where $a$ is a terminal and $x$ is a string of variables (possibly the empty string).
- It turns out that any CFG is equivalent to one in Greibach Normal Form.
- Notice unlike an s-grammar, a grammar in Greibach Normal Form is allowed to have multiple rules with the same $(A, a)$.
- Greibach Normal Form is interesting for two reasons: (1) it can be used in proofs of equivalences of CFL with those languages recognized by a certain type of automaton with a stack. (2) it also gives a slightly different upper bound on the time/space needed to parse a string in a CFG.
- To see notice, a string is generated by a a CFG in Greibach Normal Form then as each rule consumes one string letter, the derivation will be of linear length.
- The brute force algorithm on this string might still take exponential time since if we are currently reading an a, there might be several applicable rules.
- However, as all particular derivations are of linear length, the algorithm is a linear space algorithm.
- Further, if we could nondeterministically guess which rule to apply, we could find verify a derivation in linear time. This shows the context free languages are in nondeterministic linear time.
Example Converting to Greibach Normal Form

- Consider the CFG $S \rightarrow abSblaa$.
- To convert to GNF we introduce new variables $A$, $B$ with rules $A \rightarrow a$ and $B \rightarrow b$.
- The grammar
  
  $S \rightarrow aBSBlAA$
  $A \rightarrow a$
  $B \rightarrow b$

  is in GNF.