Context-free grammars

Context-free Languages
Transforming grammars

Step 1 iteration (bottom up)

- Repeat the following until $V'$ does not change
- This process will stop because there are finitely many variables & productions
  $V' = \{ A \in V : (A \rightarrow \alpha) \in P, \alpha \in (V' \cup T)^* \}$
- After some variables have been added to $V'$, we can use those variables together with the terminals to form a string $\alpha$, then if $A \rightarrow \alpha$, we add $A$ to $V'$

Remove productions with useless variables

- At the end, remove all variables not in $V'$ and productions that contain variables not in $V'$
- Call the set of remaining productions $P'$. The following is an example:
  - $S \rightarrow A | \text{Cab}$
  - $A \rightarrow SA | AaA$
  - $B \rightarrow ab | dB | bA$
  - $C \rightarrow \varepsilon | CC$
- Start: $V' = \{ B, C \}$ (use $B \rightarrow ab, C \rightarrow \varepsilon$)
- First iteration:
  - $V' = \{ B, C, S \}$ (use $S \rightarrow \text{Cab}$)
A is useless

- We see that A is useless. Remove A and all productions containing A

\[ S \rightarrow Cab \quad B \rightarrow ab \mid dB \quad C \rightarrow \varepsilon \mid CC \]

- Special note:
- If \( S \not\in V' \) then \( S \) does not derive a terminal string and \( L(G) \) must be empty
- Step 1 is an algorithm to test if \( L(G) \) is empty

more useless symbols

- We are not done, there may be more useless symbols. We find them using Step 2
- We remove anything that is not in any string derived from \( S \)
- Step 2: remove variables and terminals that do not appear in strings derived from \( S \)
- Beginning: \( V'' = \{S\}, \ T'' = \emptyset \)

Step 2 iteration (top-down)

- The iteration repeats until no new elements are added to \( V'' \) or \( T'' \)

\[ V'' = \{A \in V' : (X \rightarrow X_1 \ldots A \ldots X_n) \in P' \text{ where } X \in V'' \} \]

\[ T'' = \{a \in T : (X \rightarrow X_1 \ldots a \ldots X_n) \in P' \text{ where } X \in V'' \} \]

- Remove all remaining variables, terminals and any productions that contain symbols not in \( V'' \) or \( T'' \) the final set of productions is called \( P'' \)

example

- We had

\[ S \rightarrow Cab \quad B \rightarrow ab \mid dB \quad C \rightarrow \varepsilon \mid CC \]

\[ V'' = \{S\}, \ T'' = \emptyset \]

- use \( S \rightarrow Cab \)

\[ V'' = \{S, C\}, \ T'' = \{a, b\} \]

- neither \( C \rightarrow \varepsilon \) nor \( C \rightarrow CC \) adds any more symbols

- The productions reduce to \( S \rightarrow Cab \)

\( C \rightarrow \varepsilon \mid CC \)
Step 1 comes before Step 2

- It is vital to do Step 1 before Step 2.
- If you were to do Step 2 first, you might leave some useless symbols in the grammar.
- The textbook has an example.

Wrong order of steps

- Textbook’s example 4.8 (modified)
  \[ S \rightarrow AB \mid a \quad A \rightarrow a \mid b \quad B \rightarrow BA \]
- Apply Step 2: \( V'' = \{ S \}, \quad T'' = \emptyset \)
- From the productions, we get \( V'' = \{ S, A, B \}, \quad T'' = \{ a, b \} \)
- The grammar does not change.
- Apply Step 1: \( V' = \{ A, S \} \)
- The final grammar:
  \[ S \rightarrow a \quad A \rightarrow a \mid b \]

correct order of steps

- Apply Step 1: \( V' = \{ A, S \} \)
- The grammar \( P' \):
  \[ S \rightarrow a \quad A \rightarrow a \mid b \]
- Apply Step 2: \( V'' = \{ S \}, \quad T'' = \emptyset \)
- Iterate: \( V'' = \{ S \}, \quad T'' = \{ a \} \)
- Final grammar
  \[ S \rightarrow a \]
- We now have an algorithm to remove useless symbols.

effective procedure

- The process is also called an effective process when it implements an algorithm.
- Algorithms always terminate.
- The next part of the process is to show that \( \varepsilon \)-productions can be removed.
- The only cost is that we cannot generate \( \varepsilon \).
Textbook Theorem 4.3

- Given any CFG, say $G$, which recognizes $L$, we can modify the productions of $G$ so that there are no $\varepsilon$-productions and still have a grammar for $L - \{\varepsilon\}$
- The algorithm for this theorem has two parts:

Step 1: nullables

- Part I: find $N = \{\text{all the nullable symbols in the grammar}\}$
- A nullable symbol is a variable $A$ such that $A \Rightarrow^* \varepsilon$ ($A$ derives the empty string)
- Begin with $N = \{A \in V : (A \rightarrow \varepsilon) \in P\}$
- Iterate the following until $N$ does not change ($P'$ removes $\varepsilon$-productions):

  $$N = \{A \in V : (A \rightarrow X_1 \ldots X_n) \in P' \text{ where all } X_j \in N\}$$

example

- $S \rightarrow A \mid Cab \mid BB$
- $A \rightarrow SA \mid AaB$
- $B \rightarrow ab \mid CC \mid bA$
- $C \rightarrow \varepsilon \mid aC$
- First $N = \{C\}$ using $C \rightarrow \varepsilon$
- Next $N = \{B, C\}$ using $B \rightarrow CC$
- Next $N = \{S, B, C\}$ using $S \rightarrow BB$
- $A$ is not added to $N$
- Usually only a few of the variables are nullable but SOMETIMES all the variables are nullable

Part 2: modifying the grammar

- Part 2: add productions to take the place of $\varepsilon$-productions
- Note that $\varepsilon$-productions can be useful:
- Take the grammar

  $$F \rightarrow \text{for } (E ; G ; E) \ C$$

  $$E \rightarrow \varepsilon \mid E, E \mid \ldots$$

  $$G \rightarrow \varepsilon \mid \ldots$$

  $$C \rightarrow \ldots$$

  describing “for” in C
covering all the combinations

- Without the $\epsilon$ we have more work:
  
  \[
  F \rightarrow \text{for}(E; G; E)C \quad \text{for}(; G; E)C \\
  | \text{for}(E;; E)C \quad \text{for}(E; G;)C \\
  | \text{for}(;; E)C \quad \text{for}(; G;)C \\
  | \text{for}(E;;;)C \quad \text{for}(; ;)C
  \]

  $E \rightarrow E, E \mid ...$  
  $G \rightarrow ...$  
  $C \rightarrow ...$

start with non-empty productions

- What was just illustrated in the example is what we have to do in the grammar

- Start with $P'$ as the set of all non-empty productions, i.e. all those that do not produce $\epsilon$

- Examine every production in $P'$ and add similar productions, where nullable symbols are removed in all possible ways, without creating an $\epsilon$-production

for example

- For example, if $A \rightarrow ABCD$ is a production in $P'$ and $B$ and $C$ are nullable, then add the productions

  $A \rightarrow ABD$
  $A \rightarrow ACD$
  $A \rightarrow AD$

just do not add $\epsilon$-productions

- If $A \rightarrow ABBC$ and all the symbols $A$, $B$, $C$ are nullable, then add the productions

  $A \rightarrow BBC$  
  $A \rightarrow ABC$  
  $A \rightarrow ABB$
  $A \rightarrow BC$  
  $A \rightarrow BB$  
  $A \rightarrow AC$
  $A \rightarrow AB$  
  $A \rightarrow B$  
  $A \rightarrow C$

  - Do not add $A \rightarrow A$
  - Do not add $A \rightarrow \epsilon$
\begin{itemize}
  \item $S \rightarrow A \mid Cab \mid BB$  
  \item $A \rightarrow SA \mid AaB$  
  \item $B \rightarrow ab \mid CC \mid bA$  
  \item $C \rightarrow \epsilon \mid aC$  
  \item $\{S, B, C\}$ were nullable
  \item New productions
  \begin{itemize}
    \item $S \rightarrow A \mid Cab \mid ab \mid BB \mid B$  
    \item $A \rightarrow SA \mid AaB \mid Aa$  
    \item $B \rightarrow ab \mid CC \mid C \mid bA$  
    \item $C \rightarrow aC \mid a$
  \end{itemize}
  \item $S$ does not derive $\epsilon$ any more, e.g. $S$ did derive $a^*$ now it derives $a^+$
\end{itemize}

The scanner should have made tokens of the identifiers and numbers

\section*{Unit productions are natural}

\begin{itemize}
  \item We often see unit-productions in grammars (Pascal):
    \begin{align*}
      <stmt-seq> & \rightarrow <stmt> | \<stmt> ; <stmt-seq> \\
      <expr> & \rightarrow <term> | <term> + <expr> | <term> – <expr> \\
      <term> & \rightarrow <factor> | <factor> * <term> | <factor> / <term> \\
      <factor> & \rightarrow (<expr>) | ident | number
    \end{align*}
  \item The scanner should have made tokens of the identifiers and numbers
\end{itemize}
Removal

- Unit-productions can be removed
- Once this is done, it is easy to transform the grammar to Chomsky Normal Form (CNF)
- Removal of unit-productions makes more productions but fewer derivation steps (less deep trees)
- Before, we had the example:
  \[E \rightarrow T \mid E + T \quad T \rightarrow N \mid T * N\]
  \[N \rightarrow DN \mid D\]
  \[D \rightarrow 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9\]

Unit-production free grammar

- If we remove unit-productions but maintain the same language, we get:
  \[E \rightarrow 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | DN \mid T*N \mid E+T\]
  \[T \rightarrow 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | DN \mid T*N\]
  \[N \rightarrow 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | DN\]
  \[D \rightarrow 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9\]

Shorter derivations

- With this grammar we have
  \[E \Rightarrow E + T \Rightarrow E + T * N\]
  \[\Rightarrow E + T * 6 \Rightarrow E + 5 * 6\]
  \[\Rightarrow 3 + 5 * 6\]
- Trade-off: more productions give fewer derivations

Example

- Recall the example grammar (we added \(A \rightarrow b\) to make \(A\) useful)
  \[S \rightarrow A \mid Cab \mid BB \quad A \rightarrow SA | AaB | b\]
  \[B \rightarrow ab \mid CC \mid bA \quad C \rightarrow \epsilon \mid aC\]
- \(\{S, B, C\}\) were nullable
- New productions
  \[S \rightarrow A \mid Cab \mid ab \mid BB \mid B\]
  \[A \rightarrow SA | AaB | Aa | b\]
  \[B \rightarrow ab \mid CC \mid C | bA\]
  \[C \rightarrow aC | a\]
Unit productions

- What we see in this grammar is that we have introduced unit-productions:
- $S \rightarrow A$, $S \rightarrow B$, $B \rightarrow C$ are all unit-productions: one variable directly produces one other variable; it is still OK to produce single terminals $C \rightarrow a$

Unit-derivations come from unit-productions

- In the absence of $\epsilon$-productions, unit-derivations ONLY can come from sequences of unit-productions: $S \rightarrow B$ and $B \rightarrow C$ give $S \Rightarrow B \Rightarrow C$
- From $S \rightarrow A$, $S \rightarrow B$, $B \rightarrow C$
- We get $S \Rightarrow A$, $S \Rightarrow B \Rightarrow C$, $B \Rightarrow C$
- So $S \Rightarrow A \mid B \mid C$, $B \Rightarrow C$

Collect all non-unit productions

- Next list all non-unit productions
- Non-unit productions from the grammar discussed earlier
  - $S \rightarrow Cab \mid ab \mid BB$
  - $A \rightarrow SA \mid AaB \mid Aa \mid b$
  - $B \rightarrow ab \mid CC \mid bA$
  - $C \rightarrow aC \mid a$
Generate full grammar for same language

- For every variable $X$, list all the non-unit productions of $X$ and for every variable $Y$, such that $X \Rightarrow Y$, ADD ALL non-unit productions of $Y$ TO those of $X$:

- Given $S \Rightarrow A$, combine
  
  $S \rightarrow Cab \mid ab \mid BB$
  $A \rightarrow SA \mid AaB \mid Aa \mid b$

  giving
  
  $S \rightarrow Cab \mid ab \mid BB \mid SA \mid AaB \mid Aa \mid b$

---

Entire grammar

- After all the additions we obtain:
  
  $S \rightarrow Cab \mid ab \mid BB \mid SA \mid AaB \mid Aa \mid CC \mid bA \mid aC \mid a \mid b$
  $A \rightarrow SA \mid AaB \mid Aa \mid b$
  $B \rightarrow ab \mid CC \mid bA \mid aC \mid a$
  $C \rightarrow aC \mid a$

- DO omit duplicate productions
- Do NOT bother to remove apparent redundancies (sometimes you get completely duplicated sets of productions)

---

Another example

- $S \rightarrow SAaBC \mid aaBC \mid AA$
  $A \rightarrow ABDD \mid bDC \mid aCC$
  $B \rightarrow BAC \mid bbc \mid CS \mid \varepsilon$
  $C \rightarrow aabb \mid DBA \mid CDC$
  $D \rightarrow DSS \mid CD \mid daD \mid \varepsilon$

- Check for useless symbols, bottom-up
  
  Pass 1) $V' = \{B, C, D\}$
  Pass 2) $V' = \{S, A, B, C, D\}$

---

useless step 2 and nullables

- Check for useless symbols, top-down:
  
  $V'' = \{S\}, \ T'' = \emptyset$
  
  Pass 1) $V''' = \{S, A, B, C\}, \ T''' = \{a\}$
  Pass 2) $V''' = \{S, A, B, C, D\}, \ T''' = \{a, b, c\}$
  Pass 3) $V''' = same, \ T''' = \{a, b, c, d\}$

- Remove $\varepsilon$-productions:
- Nullables: $B \rightarrow \varepsilon, \ D \rightarrow \varepsilon$, nothing else can derive $\varepsilon$
remove all combinations of nullables

- \( S \rightarrow SAaBC \mid SAaC \mid aaBC \mid aaC \mid AA \)
- \( A \rightarrow ABDD \mid ADD \mid ABD \mid AD \mid AB \mid bDC \mid bC \mid aCC \)
- \( B \rightarrow BAC \mid AC \mid bbc \mid CS \)
- \( C \rightarrow aabb \mid DBA \mid BA \mid DA \mid A \mid CDC \mid CC \)
- \( D \rightarrow DSS \mid SS \mid CD \mid C \mid daD \mid da \)

Unit derivations \( C \rightarrow A, D \rightarrow C \), so \( D \Rightarrow A \)

Class 14 - 42 © L. Lander, 2000 42CS 573

remove unit derivations

- Use \( D \Rightarrow C \Rightarrow A \) and
- \( S \rightarrow SAaBC \mid SAaC \mid aaBC \mid aaC \mid AA \)
- \( A \rightarrow ABDD \mid ADD \mid ABD \mid AD \mid AB \mid bDC \mid bC \mid aCC \)
- \( B \rightarrow BAC \mid AC \mid bbc \mid CS \)
- \( C \rightarrow aabb \mid DBA \mid BA \mid DA \mid CDC \mid CC \)
- \( D \rightarrow DSS \mid SS \mid CD \mid C \mid daD \mid da \)

- \( D \) gets all the productions of \( A, C \)
- \( C \) gets all the productions of \( A \)

Class 14 - 43 © L. Lander, 2000 43CS 573

without unit productions

- \( S \rightarrow SAaBC \mid SAaC \mid aaBC \mid aaC \mid AA \)
- \( A \rightarrow ABDD \mid ADD \mid ABD \mid AD \mid AB \mid bDC \mid bC \mid aCC \)
- \( B \rightarrow BAC \mid AC \mid bbc \mid CS \)
- \( C \rightarrow aabb \mid DBA \mid BA \mid DA \mid CDC \mid CC \mid ABDD \mid ADD \mid ABD \mid AD \mid AB \mid bDC \mid bC \mid aCC \)
- \( D \rightarrow DSS \mid SS \mid CD \mid daD \mid da \mid ABDD \mid ADD \mid ABD \mid AD \mid AB \mid bDC \mid bC \mid aCC \mid aabb \mid DBA \mid BA \mid DA \mid CDC \mid CC \)

Class 14 - 44 © L. Lander, 2000 44CS 573

Useless symbol removal

- At this point, you must check for useless symbols again
- You will see assignments where the process of \( \varepsilon \)-production removal and unit-production removal actually makes some symbols useless
- In this example all the symbols are useful

Class 14 - 45 © L. Lander, 2000 45CS 573
The Chomsky Normal Form (CNF) for a CFG limits the form of productions. In CNF, all productions must have the form:

- either $A \rightarrow BC$
- or $A \rightarrow a$

where $A$, $B$, $C$ are any variables (productions such as $A \rightarrow AC$, $A \rightarrow AA$ are fine) and $a$ is any terminal symbol.

Conversion

Note that a CNF grammar cannot generate the empty string $\varepsilon$.

For the conversion, we assume we have removed $\varepsilon$-productions, unit-productions and all useless symbols.

What productions are problematic?

- We have to modify productions that (a) have more than a single terminal, e.g. $A \rightarrow BaCC$, $A \rightarrow ca$, $A \rightarrow aaba$ and
- (b) productions that have more than just two variables, e.g. $A \rightarrow BACC$, $A \rightarrow CCC$, $A \rightarrow BBDBCC$
Dealing with terminals

- The first step is to introduce new variables to convert every production of type (a) to a production of type (b):
  - For every terminal $t$ that appears in a production of type (a), introduce a NEW variable $V_t$ and a production $V_t \rightarrow t$

For example

- Having introduced $V_a \rightarrow a$, $V_b \rightarrow b$ and $V_c \rightarrow c$, we substitute to make type (b) productions, i.e.
  - productions
  - $A \rightarrow BaCC$, $A \rightarrow ca$, $A \rightarrow aaba$
  - become
  - $A \rightarrow BV_aCC$, $A \rightarrow V_cV_a$, $A \rightarrow V_aV_aV_bV_a$

What is left?

- Now all the productions are either of the form $A \rightarrow a$, or of the form $A \rightarrow X_1X_2...X_n$, where $X_1$, $X_2$, ..., $X_n$ are all (possibly new) variables, i.e. type (b)

It is simple to handle type (b) productions

- $A \rightarrow X_1X_2...X_n$ becomes:
  - $A \rightarrow X_1Y_1$ (so $Y_1 \rightarrow X_2...X_n$)
  - $Y_1 \rightarrow X_2Y_2$ (so $Y_2 \rightarrow X_3...X_n$)
  - $Y_2 \rightarrow X_3Y_3$ (so $Y_3 \rightarrow X_4...X_n$)
  - ...
  - $Y_{n-1} \rightarrow X_{n-1}X_n$
More new variables

- The variables $Y_1$, $Y_2$...$Y_{n-1}$ are NEW
- You can reuse NEW variables if you see the opportunity but DO NOT reuse the original variables of the grammar
- For example: $A \rightarrow BV_aCC$, $A \rightarrow V_cV_a$, $A \rightarrow V_aV_aV_bV_a$
- become: $A \rightarrow BE$, $E \rightarrow V_aF$, $F \rightarrow CC$, $A \rightarrow V_cV_a$, $A \rightarrow V_aG$, $G \rightarrow V_aH$, $H \rightarrow V_bV_a$

A previous grammar

- We had:
  $S \rightarrow Cab | ab | BB | SA | AaB | Aa / CC | bA | aC | a | b$
  $A \rightarrow SA | AaB | Aa | b$
  $B \rightarrow ab | CC | bA | aC | a$
  $C \rightarrow aC | a$

And finally the CNF

- We get:
  $S \rightarrow CDE | DE | BB | SA | ADB | AD / CC | EA / DC | a | b$
  $A \rightarrow SA | ADB | AD | b$
  $B \rightarrow DE | CC | EA / DC | a$
  $C \rightarrow DC | a$
  $D \rightarrow a$
  $E \rightarrow b$
  $F \rightarrow DE$
  $G \rightarrow DB$
BE CAREFUL

- Even though we had the production $B \rightarrow DE$, DO NOT DARE TO change
  $S \rightarrow CDE$ into
  $S \rightarrow CB$

  because we also have
  $B \rightarrow CC | EA | DC | a$

DANGER OF ERRONEOUS DERIVATIONS

- We would be adding erroneous derivations for $S$, namely
  $S \Rightarrow CCC | CEA / CDC | Ca$

- Make sure you use new variables that only have one production in the whole grammar.

The born optimizers

- Some students like to vary the substitutions used to make smaller grammars, for example, if we have:
  $A \rightarrow ABCBC | ABCB$

  $A \rightarrow AD | AF$
  $D \rightarrow EE$
  $E \rightarrow BC$
  $F \rightarrow EB$

- Or we could be more economical:
  $A \rightarrow ABCBC | ABCB$

  $A \rightarrow AD | AF$
  $D \rightarrow EE$
  $E \rightarrow BC$
  $F \rightarrow EB$

  This variability makes grading harder.
Something to avoid

- Just to illustrate that you MUST introduce new variables, consider the grammar:
  \[ S \rightarrow a \mid SASb, \; A \rightarrow b \mid AS \]
- The language generated includes \( a \) but all other strings have length at least 4 and terminate in \( b \)
- A correct CNF is
  \[
  S \rightarrow a \mid SC, \; A \rightarrow b \mid AS, \; B \rightarrow b, \\
  C \rightarrow AD, \; D \rightarrow SB
  \]

DO NOT ECONOMIZE ON SYMBOLS

- We had \( S \rightarrow a \mid SASb, \; A \rightarrow b \mid AS \)
- An erroneous conversion would be to use \( A \rightarrow b \) to substitute in the \( S \) productions:
  \[ S \rightarrow a \mid SASA, \; B \rightarrow b \]
  because that would introduce \( S \Rightarrow ababa \), which is NOT in the language
- In the erroneous grammar,
  \[ S \Rightarrow SAS \Rightarrow SASAS \Rightarrow ababa \]

Either way it is wrong

- It would be just as wrong to use \( A \rightarrow AS \) to work on the production
  \[ S \rightarrow SASb \]
- Suppose we tried
  \[ S \rightarrow a \mid SAB, \; A \rightarrow b \mid AS, \; B \rightarrow b \]
- The \( S \) could generate \( abb \), which is not in the language

The moral

- Introduce lots of new variables and if any reuse is possible, only reuse new variables