CSE 322

Construction of Reduced Grammars and Elimination of null and unit productions

Lecture #27

How to "simplify" CFGs?

Three ways to simplify/clean a CFG

(clean)

1. Eliminate *useless symbols*

(simplify)

2. Eliminate ε-productions

$A \not\Rightarrow \varepsilon$

3. Eliminate *unit productions*

$A \not\Rightarrow B$

Eliminating useless symbols

Eliminating *useless symbols*

A symbol X is *reachable* if there exists:

– $S \Rightarrow^* \alpha X \beta$

A symbol X is *generating* if there exists:

– $X \Rightarrow^* w$,
  - for some $w \in T^*$

For a symbol X to be "useful", it has to be both reachable *and* generating

- $S \Rightarrow^* \alpha X \beta \Rightarrow^* w'$, for some $w' \in T^*$

reachable generating

Algorithm to detect useless symbols

1. First, eliminate all symbols that are *not* generating

2. Next, eliminate all symbols that are *not* reachable

Is the order of these steps important,
or can we switch?

Example: Useless symbols

- S➔AB | a
- A➔ b

1. *A, S* are generating
2. *B* is *not generating* (and therefore B is useless)
3. ==> Eliminating B… (i.e., remove all productions that involve B)
   1. S➔ a
   2. A ➔ b
4. Now, A is *not reachable* and therefore is useless

5. Simplified G:
   1. S ➔ a

What would happen if you reverse the order:
i.e., test reachability before generating?

Will fail to remove: A ➔ b

$X \Rightarrow^{*} w$

Algorithm to find all generating symbols

- Given: G=(V,T,P,S)
- Basis:
  - Every symbol in T is obviously generating.
- Induction:
  - Suppose for a production A→ α, where α is generating
  - Then, A is also generating

Algorithm to find all reachable symbols

$S \Rightarrow^{*} \alpha X \beta$

- Given: G=(V,T,P,S)
- Basis:
  - S is obviously reachable (from itself)
- Induction:
  - Suppose for a production A $\rightarrow \alpha_1 \alpha_2 \ldots \alpha_k$, where A is reachable
  - Then, all symbols on the right hand side, $\{\alpha_1, \alpha_2, \ldots \alpha_k\}$ are also reachable.

Eliminating ε-productions

$A \Rightarrow \varepsilon$

X

Eliminating ε-productions

$A \rightarrow \varepsilon$

Caveat: It is *not* possible to eliminate ε-productions for languages which include ε in their word set

So we will target the grammar for the *rest of the language*

Theorem: If G=(V,T,P,S) is a CFG for a language L, then L\ {ε} has a CFG without ε-productions

Definition: A is "nullable" if A $\rightarrow^*$ ε

- If A is nullable, then any production of the form "B$\rightarrow$ CAD" can be simulated by:
  - B $\rightarrow$ CD | CAD
    - This can allow us to remove ε transitions for A

Algorithm to detect all nullable variables

- <u>Basis:</u>
  - If A➔ ε is a production in G, then A is nullable (note: A can still have other productions)
- <u>Induction:</u>
  - If there is a production B➔ $C_1C_2…C_k$, where *every* C_i is nullable, then B is also nullable

Eliminating ε-productions

<u>Given:</u> G=(V,T,P,S)

<u>Algorithm:</u>

1. Detect all nullable variables in G
2. Then construct $G_1=(V,T,P_1,S)$ as follows:
   i. For each production of the form: $A \rightarrow X_1X_2\ldots X_k$, where $k \geq 1$, suppose ***m*** out of the ***k*** X_i's are nullable symbols
   ii. Then G_1 will have **2^m** versions for this production
      i. i.e, all combinations where each X_i is either present or absent
   iii. Alternatively, if a production is of the form: $A \rightarrow \varepsilon$, then remove it

Example: Eliminating ε-productions

- Let L be the language represented by the following CFG G:
  i. S ➔ AB
  ii. A ➔ aAA | ε
  iii. B ➔ bBB | ε

Goal: To construct G1, which is the grammar for L-{ε}

- Nullable symbols: {A, B}

- G_1 can be constructed from G as follows:
  - B ➔ b | bB | bB | bBB
    - ==> B ➔ b | bB | bBB
  - Similarly, A ➔ a | aA | aAA
  - Similarly, S ➔ A | B | AB

- Note: $L(G) = L(G_1) \cup \{\varepsilon\}$

Simplified grammar

G_1:
- S ➔ A | B | AB
- A ➔ a | aA | aAA
- B ➔ b | bB | bBB

\+

- S ➔ ε

Eliminating unit productions

A => B

B has to be a variable

What's the point of removing unit transitions ?

Will save #substitutions

E.g.,

before	after
A=>B | …	A=>xxx | yyy | zzz | …
B=>C | …	B=> xxx | yyy | zzz | …
C=>D | …	C=> xxx | yyy | zzz | …
D=>xxx | yyy | zzz	D=>xxx | yyy | zzz

Eliminating unit productions

A ➔ B

- Unit production is one which is of the form A➔ B, where both A & B are variables
- E.g.,
  1. E ➔ T | E+T
  2. T ➔ F | T*F
  3. F ➔ I | (E)
  4. I ➔ a | b | Ia | Ib | I0 | I1
  - How to eliminate unit productions?
    - Replace E➔ T with E ➔ F | T*F
    - Then, upon recursive application wherever there is a unit production:
      - E➔ F | T*F | E+T (substituting for T)
      - E➔ I | (E) | T*F| E+T (substituting for F)
      - E➔ a | b | Ia | Ib | I0 | I1 | (E) | T*F | E+T (substituting for I)
      - Now, E has no unit productions
    - Similarly, eliminate for the remainder of the unit productions

The **Unit Pair Algorithm**: to remove unit productions

- Suppose $A \rightarrow B_1 \rightarrow B_2 \rightarrow \ldots \rightarrow B_n \rightarrow \alpha$
- Action: Replace all intermediate productions to produce α directly
  - i.e., $A \rightarrow \alpha$; $B_1 \rightarrow \alpha$; … $B_n \rightarrow \alpha$;

Definition: (A,B) to be a ***"unit pair"*** if $A \rightarrow^* B$

- We can find all unit pairs inductively:
  - Basis: Every pair (A,A) is a unit pair (by definition). Similarly, if $A \rightarrow B$ is a production, then (A,B) is a unit pair.
  - Induction: If (A,B) and (B,C) are unit pairs, and $A \rightarrow C$ is also a unit pair.

The Unit Pair Algorithm: to remove unit productions

Input: $G=(V,T,P,S)$

Goal: to build $G_1=(V,T,P_1,S)$ devoid of unit productions

Algorithm:

1. Find all unit pairs in G
2. For each unit pair (A,B) in G:
    1. Add to P_1 a new production $A \rightarrow \alpha$, for every $B \rightarrow \alpha$ which is a *non-unit* production
    2. If a resulting production is already there in P, then there is no need to add it.

Example: eliminating unit productions

G:
1. E ➔ T | E+T
2. T ➔ F | T*F
3. F ➔ I | (E)
4. I ➔ a | b | Ia | Ib | I0 | I1

Unit pairs	*Only non-unit productions to be added to P_1*
(E,E)	E ➔ E+T
(E,T)	E ➔ T*F
(E,F)	E ➔ (E)
(E,I)	E ➔ a|b|Ia | Ib | I0 | I1
(T,T)	T ➔ T*F
(T,F)	T ➔ (E)
(T,I)	T ➔ a|b| Ia | Ib | I0 | I1
(F,F)	F ➔ (E)
(F,I)	F ➔ a| b| Ia | Ib | I0 | I1
(I,I)	I ➔ a| b | Ia | Ib | I0 | I1

G_1:
1. E ➔ E+T | T*F | (E) | a| b | Ia | Ib | I0 | I1
2. T ➔ T*F | (E) | a| b | Ia | Ib | I0 | I1
3. F ➔ (E) | a| b | Ia | Ib | I0 | I1
4. I ➔ a | b | Ia | Ib | I0 | I1

Putting all this together…

- Theorem: If G is a CFG for a language that contains at least one string other than ε, then there is another CFG G_1, such that $L(G_1)=L(G) - \varepsilon$, *and* G_1 has:
  - no ε -productions
  - no unit productions
  - no useless symbols

- Algorithm:

Step 1) eliminate ε -productions
Step 2) eliminate unit productions
Step 3) eliminate useless symbols

Again,
the order is
important!

Why?

PROBLEM

Consider the grammar G whose productions are $S \rightarrow aS \mid AB$, $A \rightarrow \Lambda$, $B \rightarrow \Lambda$, $D \rightarrow b$. Construct a grammar G_1 without null productions generating $L(G) - \{\Lambda\}$.

Step 1 *Construction of the set W of all nullable variables:*

$$W_1 = \{A_1 \in V_N \mid A_1 \to \Lambda \text{ is a production in } G\}$$

$$= \{A, B\}$$

$$W_2 = \{A, B\} \cup \{S\} \text{ as } S \to AB \text{ is a production with } AB \in W_1^*$$

$$= \{S, A, B\}$$

$$W_3 = W_2 \cup \emptyset = W_2$$

Thus,

$$W = W_2 = \{S, A, B\}$$

Step 2 *Construction of P':*

(i) $D \to b$ is included in P'.

(ii) $S \to aS$ gives rise to $S \to aS$ and $S \to a$.

(iii) $S \to AB$ gives rise to $S \to AB$, $S \to A$ and $S \to B$.

(*Note:* We cannot erase both the nullable variables A and B in $S \to AB$ as we will get $S \to \Lambda$ in that case.)

Hence the required grammar without null productions is

$$G_1 = (\{S, A, B, D\}, \{a, b\}, P, S)$$

where P' consists of

$$D \to b, \ S \to aS, \ S \to AB, \ S \to a, \ S \to A, \ S \to B$$