

## **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  $\varepsilon$ -productions

$$A \Rightarrow \epsilon$$

3. Eliminate unit productions



## Eliminating useless symbols

## Eliminating useless symbols



A symbol X is <u>reachable</u> if there exists:

$$-$$
 S  $\rightarrow$ \*  $\alpha$  X  $\beta$ 

A symbol X is *generating* if there exists:

- X →\* w,
  - for some  $w \in T^*$

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

• S  $\rightarrow$ \*  $\alpha \times \beta \rightarrow$ \* w', for some w'  $\in T$ \*

reachable generating

## Algorithm to detect useless symbols

First, eliminate all symbols that are not generating

Next, eliminate all symbols that are not reachable

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

## Example: Useless symbol

- 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





### Algorithm to find all generating symbols

- <u>Given:</u> G=(V,T,P,S)
- Basis:
  - Every symbol in T is obviously generating.
- Induction:
  - Suppose for a production  $A \rightarrow \alpha$ , where  $\alpha$  is generating
  - Then, A is also generating



 $S \rightarrow^* \alpha X \beta$ 

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



## Eliminating ε-productions





## Eliminating ε-productions



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

So we will target the grammar for the <u>rest</u> of the language

<u>Theorem:</u> If G=(V,T,P,S) is a CFG for a language L, then L\  $\{\mathcal{E}\}$  has a CFG without  $\mathcal{E}$ -productions

#### <u>Definition:</u> A is "nullable" if $A \rightarrow * \mathcal{E}$

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



## Algorithm to detect all nullable variables

#### • Basis:

- If  $A \rightarrow \epsilon$  is a production in G, then A is nullable (note: A can still have other productions)

#### • Induction:

– If there is a production  $B \rightarrow C_1C_2...C_k$ , where *every*  $C_i$  is nullable, then B is also nullable



## Eliminating ε-productions

Given: G=(V,T,P,S)

#### Algorithm:

- 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:  $A \rightarrow X_1 X_2 ... X_k$ , where  $k \ge 1$ , 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<sub>i</sub> is either present or absent
  - iii. Alternatively, if a production is of the form:  $A \rightarrow \epsilon$ , then remove it



Simplified

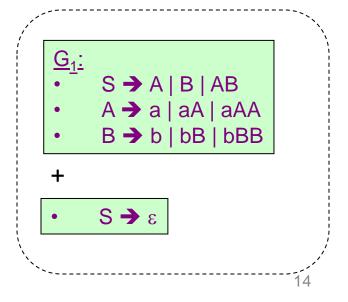
grammar

### Example: Eliminating ε-productions

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

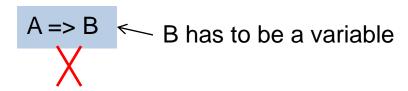
Goal: To construct G1, which is the grammar for L- $\{\varepsilon\}$ 

- Nullable symbols: {A, B}
- G₁ can be constructed from G as follows:
  - B → b | bB | bB | bBB
    - - ==> B → b | bB | bBB
- Similarly,  $A \rightarrow a \mid aA \mid aAA$ 
  - - Similarly,  $S \rightarrow A \mid B \mid AB$
- Note:  $L(G) = L(G_1) \cup \{\epsilon\}$





## Eliminating unit productions



What's the point of removing unit transitions?

Will save #substitutions

15

## Eliminating unit productions

### $A \rightarrow B$

- Unit production is one which is of the form A→ B, where both A & B are variables
- E.g.,
  - E → T | E+T 1.
  - T → F | T\*F
  - 3.  $F \rightarrow || (E)$
  - I → a | b | Ia | Ib | IO | I1
  - How to eliminate unit productions?
    - Replace  $E \rightarrow T$  with  $E \rightarrow F \mid T^*F$
    - Then, upon recursive application wherever there is a unit production:
      - E → F | T\*F | E+T (substituting for T)
      - E→ | | (E) | T\*F| E+T (substituting for F) (substituting for I)
      - E→ a | b | la | lb | l0 | l1 | (E) | T\*F | E+T
      - Now, E has no unit productions
    - Similarly, eliminate for the remainder of the unit productions





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

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

<u>Definition:</u> (A,B) to be a "unit pair" if A→\*B

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

## The Unit Pair Algorithm: to remove unit productions

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

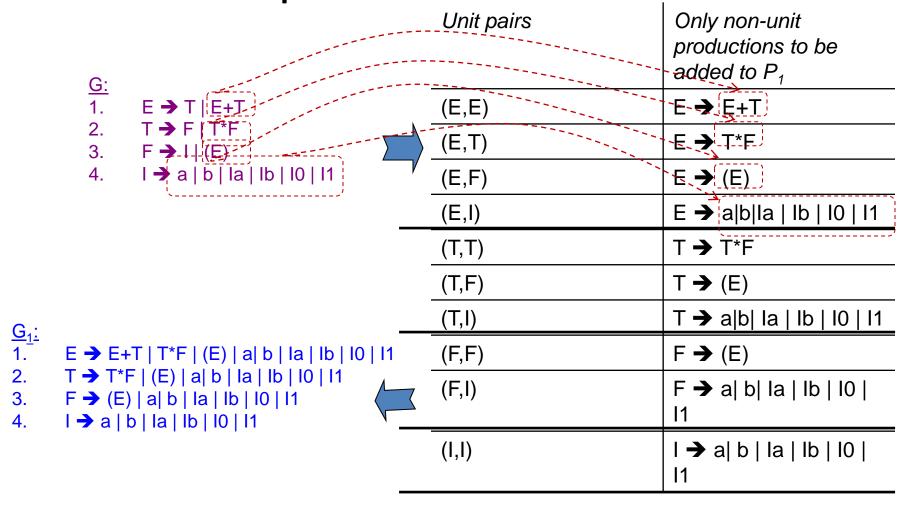
<u>Goal</u>: to build  $G_1=(V,T,P_1,S)$  devoid of unit productions

#### 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





- Theorem: If G is a CFG for a language that contains at least one string other than  $\varepsilon$ , 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  $\varepsilon$  -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  $S \to aS \mid AB$ ,  $A \to \Lambda$ ,  $B \to \Lambda$ ,  $D \to b$ . Construct a grammar  $G_1$  without null productions generating  $L(G) - \{\Lambda\}$ .



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

$$W_1 = \{A_1 \in V_N | A_1 \rightarrow A \text{ is a production in } G\}$$

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

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

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

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

Thus.

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



#### **Step 2** Construction of P':

- (i)  $D \rightarrow b$  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

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