



CSCI 742 - Compiler Construction

Lecture 11
Chomsky Normal Form (CNF)
Instructor: Hossein Hojjat

February 15, 2017

Recap: Chomsky Normal Form (CNF)

A CFG is in Chomsky Normal Form if each rule is of the form

$$A \rightarrow BC$$

$$A \rightarrow a$$

where

- a is any terminal
- A, B, C are non-terminals
- B, C cannot be start variable

We allow the rule $S \rightarrow \epsilon$ if $\epsilon \in L$

Conversion to Chomsky Normal Form (CNF)

Steps: (not in the optimal order)

1. remove unproductive non-terminals
2. remove unreachable non-terminals
3. remove ϵ -production rules $X \rightarrow \epsilon$ (X is not start non-terminal)
4. remove single non-terminal productions (unit production rules)
($X \rightarrow Y$)
5. reduce arity of every production to less than two
6. make terminals occur alone on right-hand side

(1) Unproductive Non-terminals

- Consider the following grammar with start non-terminal “stmt”

```
stmt → identifier := identifier
      | while (expr) stmt
      | if (expr) stmt else stmt
expr  → term + term | term - term
term  → factor * factor
factor → (expr)
```

- There is no derivation of a sequence of tokens from **expr**
- Every derivation step of **expr** has at least one **expr**, **term**, or **factor**
- If a non-terminal cannot derive sequence of tokens we call it **unproductive**

(1) Unproductive Non-terminals

Productive Non-terminals

- Productive non-terminals are obtained using these two rules (what remains is unproductive)
 - 1) Terminals are productive
 - 2) If $A \rightarrow \alpha$ is a production rule and each non-terminal symbols of α is productive then A is also productive (α can also be ϵ)

Remove Unproductive Non-terminals

- Remove all production rules in which an unproductive non-terminal appears either on the left or the right

Exercise

Question:

- Remove all the unproductive non-terminals from the following grammar.

$$S \rightarrow B \mid AC$$

$$B \rightarrow aAa$$

$$A \rightarrow \epsilon$$

$$C \rightarrow cC \mid DA$$

$$D \rightarrow C$$

Exercise

Question:

- Remove all the unproductive non-terminals from the following grammar.

$$S \rightarrow B \mid AC$$

$$B \rightarrow aAa$$

$$A \rightarrow \epsilon$$

$$C \rightarrow cC \mid DA$$

$$D \rightarrow C$$

Answer:

$$S \rightarrow B$$

$$B \rightarrow aAa$$

$$A \rightarrow \epsilon$$

(2) Unreachable non-terminals

- Consider the following grammar with start non-terminal “program”

program \rightarrow stmt | stmt program

stmt \rightarrow assignment | whileStmt

assignment \rightarrow expr = expr

ifStmt \rightarrow if (expr) stmt else stmt

whileStmt \rightarrow while (expr) stmt

- No way to reach non-terminal “**ifStmt**” from “program”

(2) Unreachable non-terminals

Reachable Non-terminals

- Reachable non-terminals are obtained using these two rules (what remains is unreachable)
 - 1) Starting non-terminal is reachable
 - 2) If $A \rightarrow \alpha$ is a production rule and A is reachable, each non-terminal symbols of α is also reachable

Remove Unreachable Non-terminals

- Remove all production rules in which an unreachable non-terminal appears either on the left or the right

(3) Removing ϵ -Production Rules

- Ensure only top-level non-terminal can be nullable

Original Grammar

```
program → stmtSeq
stmtSeq → stmt | stmt ; stmtSeq
stmt → "" | assignment
      | whileStmt | blockStmt
blockStmt → { stmtSeq }
assignment → expr = expr
whileStmt → while (expr) stmt
expr → identifier
```

Grammar after removing ϵ -rules

```
program → "" | stmtSeq
stmtSeq → stmt | stmt ; stmtSeq
        | ; stmtSeq | stmt ; | ;
stmt → assignment
     | whileStmt | blockStmt
blockStmt → { stmtSeq } | {}
assignment → expr = expr
whileStmt → while (expr) stmt
whileStmt → while (expr)
expr → identifier
```

- **Definition:** Variable X is nullable if $X \Rightarrow^* \epsilon$
- Rules to compute the nullable variables of a grammar:
 - 1) If $A \rightarrow \epsilon$ is a production rule then A is *nullable*
 - 2) If $B \rightarrow X_1X_2 \cdots X_n$ is a production rule and all the X_i are *nullable* then B is also *nullable*

(3) Removing ϵ -Production Rules

- Compute the set of *nullable* non-terminals
- For each rule $A \rightarrow X_1 \cdots X_n$ add **all** production rules that can be formed by eliminating **any** combination of *nullable* X_i 's
- Repeat the above step for the newly added rules
- Remove all rules with empty right-hand sides
- If starting symbol S was nullable, then introduce a new start symbol S' instead, and add rule $S' \rightarrow S \mid \epsilon$
- Note: number of added rules for $A \rightarrow X_1 \cdots X_n$ is $O(2^k)$ (where k is the number of *nullable* X_i 's)

(3) Removing ϵ -Production Rules

- Since `stmtSeq` is *nullable*, the rule
`blockStmt` \rightarrow `{ stmtSeq }`
gives
`blockStmt` \rightarrow `{ stmtSeq }` | `{ }`
- Since `stmtSeq` and `stmt` are *nullable*, the rule
`stmtSeq` \rightarrow `stmt` | `stmt ; stmtSeq`
gives
`stmtSeq` \rightarrow `stmt` | `stmt ; stmtSeq`
| `; stmtSeq` | `stmt ;` | `;`

Question:

- 1) Remove the ϵ production rules from the following grammar.
- 2) Remove unproductive non-terminals after step 1.

$$S \rightarrow ABC$$

$$B \rightarrow CAb \mid b$$

$$C \rightarrow ASD \mid AD$$

$$D \rightarrow CaA \mid \epsilon$$

$$A \rightarrow \epsilon$$

Exercise

Question:

- 1) Remove the ϵ production rules from the following grammar.
- 2) Remove unproductive non-terminals after step 1.

$$S \rightarrow ABC$$

$$B \rightarrow CAb \mid b$$

$$C \rightarrow ASD \mid AD$$

$$D \rightarrow CaA \mid \epsilon$$

$$A \rightarrow \epsilon$$

Answer:

After removing ϵ rules:

$$S \rightarrow ABC \mid AB \mid B \mid BC$$

$$B \rightarrow CAb \mid Ab \mid b \mid Cb$$

$$C \rightarrow ASD \mid AD \mid AS \mid S \mid SD \mid A \mid D$$

$$D \rightarrow CaA \mid Ca \mid a \mid aA$$

(4) Eliminating unit productions

- Single production rule is of the form

$$X \rightarrow Y$$

where X, Y are non-terminals

```
program → stmtSeq
stmtSeq → stmt
         | stmt ; stmtSeq
stmt → assignment | whileStmt
assignment → expr = expr
whileStmt → while (expr) stmt
```


(4) Eliminating unit productions

- If there is a unit production $X \rightarrow Y$ put an edge (X, Y) into graph
- If there is a path from Y to Z in the graph, and there is rule $Z \rightarrow X_1X_2 \cdots X_n$ then add rule $Y \rightarrow X_1X_2 \cdots X_n$

At the end, remove all unit productions

(4) Eliminating unit productions

program \rightarrow stmtSeq
stmtSeq \rightarrow stmt | stmt ; stmtSeq
stmt \rightarrow assignment | whileStmt
assignment \rightarrow expr = expr
whileStmt \rightarrow while (expr) stmt

After removing unit productions:

program \rightarrow expr = expr | while (expr) stmt
| stmt ; stmtSeq
stmtSeq \rightarrow expr = expr | while (expr) stmt
| stmt ; stmtSeq
stmt \rightarrow expr = expr | while (expr) stmt
assignment \rightarrow expr = expr
whileStmt \rightarrow while (expr) stmt

(5) Reducing Arity

- No more than 2 non-terminals on RHS

`stmt` \rightarrow `while` (`expr`) `stmt`

- becomes

`stmt` \rightarrow `while` `stmt`₁

`stmt`₁ \rightarrow (`stmt`₂

`stmt`₂ \rightarrow `expr` `stmt`₃

`stmt`₃ \rightarrow `)stmt`

(6) A non-terminal for each terminal

$\text{stmt} \rightarrow \text{while } (\text{expr}) \text{ stmt}$

- becomes

$\text{stmt} \rightarrow N_{\text{while}} \text{stmt}_1$

$\text{stmt}_1 \rightarrow N_{(} \text{stmt}_2$

$\text{stmt}_2 \rightarrow \text{expr} \text{stmt}_3$

$\text{stmt}_3 \rightarrow N_{)} \text{stmt}$

$N_{\text{while}} \rightarrow \text{while}$

$N_{(} \rightarrow ($

$N_{)} \rightarrow)$

Order of steps in conversion to CNF

1. remove unproductive non-terminals (optional)
2. remove unreachable non-terminals (optional)
3. make terminals occur alone on right-hand side
4. reduce arity of every production to ≤ 2
5. remove epsilons
6. remove unit productions $X \rightarrow Y$
7. unproductive non-terminals
8. unreachable non-terminals