Conversion to Chomsky Normal Form (CNF)

Steps: (not in the optimal order)

- remove unproductive symbols
- remove unreachable symbols
- remove epsilons (no non-start nullable symbols)
- remove single non-terminal productions (unit productions) X::=Y
- reduce arity of every production to less than two
- make terminals occur alone on right-hand side

1) Unproductive non-terminals

What is funny about this grammar:

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

In every step will have at least one expr, term, or factor

If it cannot derive sequence of tokens we call it *unproductive*

1) Unproductive non-terminals

Productive symbols are obtained using these two rules (what remains is unproductive)

- Terminals are productive
- If $X ::= s_1\ s_2 \dots s_n$ is a rule and each s_i is productive then X is productive

Delete unproductive symbols.

The language recognized by the grammar will not change

2) Unreachable non-terminals

What is funny about this grammar with start symbol 'program'

program ::= stmt | stmt program
stmt ::= assignment | whileStmt
assignment ::= expr = expr
ifStmt ::= if (expr) stmt else stmt
whileStmt ::= while (expr) stmt
expr ::= identifier

No way to reach symbol 'ifStmt' from 'program'

Can we formulate rules for reachable symbols ?

2) Unreachable non-terminals

Reachable terminals are obtained using the following rules (the rest are unreachable)

- starting non-terminal is reachable (program)
- If X::= s_1 s_2 … s_n is rule and X is reachable then every non-terminal in s_1 s_2 … s_n is reachable

Delete unreachable nonterminals and their productions

3) Removing Empty Strings

Ensure only top-level symbol can be nullable

program ::= stmtSeq
stmtSeq ::= stmt | stmt ; stmtSeq
stmt ::= “” | assignment | whileStmt | blockStmt
blockStmt ::= { stmtSeq }
assignment ::= expr = expr
whileStmt ::= while (expr) stmt
expr ::= identifier

How to do it in this example?

3) Removing Empty Strings - Result

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

3) Removing Empty Strings - Algorithm

$O(2^n)$

3) Removing Empty Strings

- Since stmtSeq is nullable, the rule

  blockStmt ::= { stmtSeq }

  gives

  blockStmt ::= { stmtSeq } | { }
- Since stmtSeq and stmt are nullable, the rule

  stmtSeq ::= stmt | stmt ; stmtSeq

  gives

  stmtSeq ::= stmt | stmt ; stmtSeq
  | ; stmtSeq | stmt ; | ;

4) Eliminating unit productions

- Single production is of the form

  X ::=Y

where X,Y are non-terminals

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

4) Unit Production Elimination Algorithm

- If there is a unit production

  X ::=Y put an edge (X,Y) into graph

- If there is a path from X to Z in the graph, and there is rule $Z ::= s_1\ s_2 \ldots s_n$ then add rule

$$X ::= s_1\ s_2 \ldots s_n$$

At the end, remove all unit productions.

4) Eliminate unit productions - Result

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

5) Reducing Arity: No more than 2 symbols on RHS

stmt ::= while (expr) stmt

becomes

$stmt$ ::= while $stmt_1$

$stmt_1$ ::= ($stmt_2$

$stmt_2$ ::= expr $stmt_3$

$stmt_3$ ::=) stmt

6) A non-terminal for each terminal

$stmt ::= while\ (expr)\ stmt$

becomes

$stmt ::= N_{while}\ stmt_1$

$stmt_1 ::= N_(\ stmt_2$

$stmt_2 ::= expr\ stmt_3$

$stmt_3 ::= N_)\ stmt$

$N_{while} ::= while$

$N_(::= ($

$N_) ::=)$

Order of steps in conversion to CNF

1. remove unproductive symbols (optional)
2. remove unreachable symbols (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::=Y
7. unproductive symbols
8. unreachable symbols

- What if we swap the steps 4 and 5 ?
  - Potentially exponential blow-up in the # of productions

Ordering of Unreachable / Unproductive symbols

First Unreachable then Unproductive

S := B C | “”
C := D
D := a
R := r

S := B C | “”
C := D
D := a

S := “”
C := D
D := a

First Unproductive then Unreachable

S := B C | “”
C := D
D := C
R := r

S := “”
C := D
D := a
R := r

S := “”

Alternative

We need not go all the way to Chomsky form

it is possible to directly parse arbitrary grammar

Key steps: (not in the optimal order)

- reduce arity of every production to less than two (otherwise, worse than cubic in string input size)

Can be less efficient in grammar size, but still works

A well-known algorithm for arbitrary grammars:

Earley's parsing algorithm