COMP2022
Models of Computation

Chomsky Normal Form and Parsing

Sasha Rubin

August 29, 2024

THE UNIVERSITY OF SYDNEY

Agenda

1. Chomsky Normal Form (CNF) for CFGs
2. Parsing CFGs

Normal forms

An object can have more than one representation. A normal form is a "standard" or "canonical" representation of objects.

Example, lines in high-school geometry have normal forms:

- General form $Ax + By = C$
- Gradient-intercept form $y = mx + c$

You can often think of converting an object into a normal form as a "preprocessing" step that makes it easier to write and understand algorithms.

You will see important normal forms in Databases and Logic.

Chomsky Normal Form

Definition. A grammar is in Chomsky Normal Form (CNF) if every rule is in of one of these forms:

1. $A \to BC$ (A, B, C are any variables, except that neither B nor C is the start variable)
2. $A \to a$ (A is any variable and a is a terminal)
3. In addition, we permit $S \to \varepsilon$ where S is the start variable.

Note.

- We disallow rules like $A \to 0A1$ and $A \to B$ and $A \to ab$
- Parse trees of a grammar in CNF are binary trees (i.e., every internal node has 1 or 2 children).

$$T \to aTb \mid \epsilon$$

$$\begin{aligned} S &\to AX \mid \epsilon \\ T &\to AX \\ X &\to TB \mid b \\ A &\to a \\ B &\to b \end{aligned}$$

How can we turn a CFG into CNF?

Let's give names to some of the rules we want to eliminate.

1. rule of the form $A \to \epsilon$ is called an epsilon rule.
2. rule of the form $A \to B$ is called a unit rule.

Given a grammar G:

1. We transform G into an equivalent grammar G' that has no epsilon rules except perhaps $S \to \epsilon$.
2. We transform G' into an equivalent grammar G'' that also has no unit rules.

We do each of these by removing some rules and adding some rules...

Remove epsilon rules: idea

The idea is to **inline:**

$$\text{if } B \to uAv \text{ and } A \overset{+}{\Rightarrow} \epsilon \text{ then also } B \to uv$$

1. Find all variables $A \overset{+}{\Rightarrow} \epsilon$ ('nullable variable') — see tutorial
2. Remove from R all rules of the form $A \to \epsilon$
3. If $B \to uAv$ is in R and $A \overset{+}{\Rightarrow} \epsilon$ then add the rule $B \to uv$ to R (as long as $uv \neq \epsilon$)
4. Repeat the previous step until no new rules are added.
5. Add $S \to \epsilon$ to R if $S \overset{+}{\Rightarrow} \epsilon$

Note: This procedure may add unit rules.

Remove epsilon rules: example

$$S \to AB \mid \epsilon$$
$$A \to aA \mid \epsilon$$
$$B \to bB \mid \epsilon$$

$$S \to AB \mid A \mid B \mid \epsilon$$
$$A \to aA \mid a$$
$$B \to bB \mid b$$

Removing unit rules: idea

The idea is to **inline:**

$$\text{If } B \to u \text{ and } A \overset{+}{\Rightarrow} B \text{ then also } A \to u$$

1. Find all pairs A, B such that $A \overset{+}{\Rightarrow} B$ — see tutorial
2. Remove from R all rules of the form $A \to B$
3. If $B \to u$ is in R and $A \overset{+}{\Rightarrow} B$ then add $A \to u$ to R
4. Repeat the previous step until no new rules are added.

Removing unit rules: example

$$S \to AB \mid A \mid B \mid \epsilon$$
$$A \to aA \mid a$$
$$B \to bB \mid b$$

$$S \to AB \mid aA \mid a \mid bB \mid b \mid \epsilon$$
$$A \to aA \mid a$$
$$B \to bB \mid b$$

Chomsky Normal Form

Theorem
Every context-free language is generated by a grammar in CNF.

In the next slides, we will give a 5-step algorithm to do this:

1. START: Eliminate the start variable from the RHS of all rules
2. TERM: Eliminate rules with terminals (except of form $A \to a$)
3. BIN: Eliminate rules with more than two variables
4. EPSILON: Eliminate epsilon rules (except $S \to \epsilon$)
5. UNIT: Eliminate unit rules

Chomsky Normal Form: algorithm

START: Eliminate the start variable from the RHS of all rules

---

- Add the new start variable S and the rule $S \to T$ where T was the old start variable.

Chomsky Normal Form: algorithm

TERM: Eliminate rules with terminals (except of form $A \to a$)

- Replace every terminal a on the RHS of a rule (that is not of the form $A \to a$) by the new variable N_a.
- For each such terminal a create the new rule $N_a \to a$.

Chomsky Normal Form: algorithm

BIN: Eliminate rules with more than two variables

For every rule of the form $A \to DEFGH$, say, delete it and create new variables A_1, A_2, A_3 and add rules:

$$A \to DA_1$$
$$A_1 \to EA_2$$
$$A_2 \to FA_3$$
$$A_3 \to GH$$

Chomsky Normal Form: algorithm

EPSILON: Eliminate epsilon rules (except $S \to \epsilon$)

- Use the procedure from before.
- This will not mess up the results of START, TERM, BIN since it only removes symbols from the right-hand-side of some rules. So:
  - it will not add the start variable to the RHS of a rule
  - it will not add rules with terminals (since the only rules with terminals are of the form $A \to a$)
  - it will not add rules with more than two variables
- It may introduce unit rules.

Chomsky Normal Form: algorithm

UNIT: Eliminate unit rules

- Use the procedure from before.
- This will not mess up START,TERM, BIN since it only adds rules that change the left-hand-sides of some rules.
- It will also not add epsilon rules. Why? because to add $A \to \epsilon$ we need that $S \to \epsilon$ (since this is the only possible epsilon rule) and $A \stackrel{+}{\Rightarrow} S$, but because of START we know that S does not occur on the right-hand-side of any rule.

Chomsky Normal Form: example

$$T \to aTb \mid \epsilon$$

1. START: Eliminate start variable from the RHS of all rules:

$$S \to T$$
$$T \to aTb \mid \epsilon$$

Chomsky Normal Form: example

$$S \to T$$
$$T \to aTb \mid \epsilon$$

2. TERM: Eliminate rules with terminals (except form $A \to a$):

$$S \to T$$
$$T \to ATB \mid \epsilon$$
$$A \to a$$
$$B \to b$$

Chomsky Normal Form: example

$$
\begin{aligned}
S &\to T \\
T &\to ATB \mid \epsilon \\
A &\to a \\
B &\to b
\end{aligned}
$$

3. BIN: Eliminate rules with more than two variables:

$$
\begin{aligned}
S &\to T \\
T &\to AX \mid \epsilon \\
X &\to TB \\
A &\to a \\
B &\to b
\end{aligned}
$$

Chomsky Normal Form: example

$$
\begin{aligned}
S &\to T \\
T &\to AX \mid \epsilon \\
X &\to TB \\
A &\to a \\
B &\to b
\end{aligned}
$$

4. EPSILON: Eliminate epsilon rule $T \to \varepsilon$

$$
\begin{aligned}
S &\to T \mid \epsilon \\
T &\to AX \\
X &\to TB \mid B \\
A &\to a \\
B &\to b
\end{aligned}
$$

Chomsky Normal Form: example

$$
\begin{aligned}
S &\to T \mid \epsilon \\
T &\to AX \\
X &\to TB \mid B \\
A &\to a \\
B &\to b
\end{aligned}
$$

5. UNIT: Eliminate unit rules (first $S \to T$, then $X \to B$)

$$
\begin{aligned}
S &\to AX \mid \epsilon \\
T &\to AX \\
X &\to TB \mid b \\
A &\to a \\
B &\to b
\end{aligned}
$$

Chomsky Normal Form: example

All done!

$$
\begin{aligned}
S &\to AX \mid \epsilon \\
T &\to AX \\
X &\to TB \mid b \\
A &\to a \\
B &\to b
\end{aligned}
$$

COMP2022|2922
Models of Computation

CYK Algorithm for Parsing CFGs in CNF

Sasha Rubin

August 29, 2024

THE UNIVERSITY OF SYDNEY

- A context-free grammar (CFG) generates strings by rewriting.
- Today we will see how to tell if a given context-free grammar (CFG) generates a given string.
- This basic problem is solved by compilers and parsers.

Membership problem for CFG: Given a CFG G and string w decide if G generates w.

Another related problem is **Parsing problem for CFG:** Given a CFG G and string w produce a parse-tree for w or say there is none.

Chomsky: Can you think of an algorithm that decides if w is generated by a CFG G?

Student: Sure, just search through all derivations until you find one that derives w. E.g., try all i-step derivations for $i = 1, 2, 3, etc.$.

C: But what if w is not generated by G, how will your algorithm deduce this?

S: Right, I need a stopping condition...

Question: If G is in CNF then every non-empty string $w \in L(G)$ can be derived in at most how many steps?

1. 1000
2. $|w|$
3. $2\log(|w|)$
4. $2|w| - 1$

S: Actually, we can reason that every derivation of w must take this many steps. So we can just search through all such derivations and see if any produces w.

C: Right. How many derivations with N steps are there?

S: Well, in general, at least 2^N, because at each step we may have two or more possible rules to fire for the (leftmost) variable... This means my procedure runs in exponential time, which is slooooow.

Today we will see a polynomial time algorithm.[1]

[1] Pseudocode from "Introduction to the theory of computation" by Michael Sipser, 3rd edition, Theorem 7.16.

Define a procedure *table*(V, i, j) that decides if G can derive $w_i \cdots w_j$ starting with the nonterminal V.

Then we just check *table*$(S, 0, len(w) - 1)$.[2]

We can also write $V \in$ *table*(i, j) instead of *table*(V, i, j).

We can draw *table* as a trianglular table..

[2] If $w = \epsilon$ we just check if $S \to \epsilon$ is a rule.

$$S \to AB \mid AX \mid \epsilon$$
$$T \to AB \mid AX$$
$$X \to TB$$
$$A \to a$$
$$B \to b$$

S,T			
∅	X		
∅	S,T	∅	
A	A	B	B

$w = aabb$

(0,3)			
(0,2)	(1,3)		
(0,1)	(1,2)	(2,3)	
(0,0)	(1,1)	(2,2)	(3,3)

E.g., the only variable that derives $w[1,3] = abb$ is X
E.g., no variable derives $w[0,2] = aab$

Fill in the "base" diagonal by looking at rules of the form $C \to c$.

$$S \to AB \mid AX \mid \epsilon$$
$$T \to AB \mid AX$$
$$X \to TB$$
$$A \to a$$
$$B \to b$$

?	?	?	?

$w = aabb$

(0,3)			
(0,2)	(1,3)		
(0,1)	(1,2)	(2,3)	
(0,0)	(1,1)	(2,2)	(3,3)

$$S \to AB \mid AX \mid \epsilon$$
$$T \to AB \mid AX$$
$$X \to TB$$
$$A \to a$$
$$B \to b$$

A	A	B	B

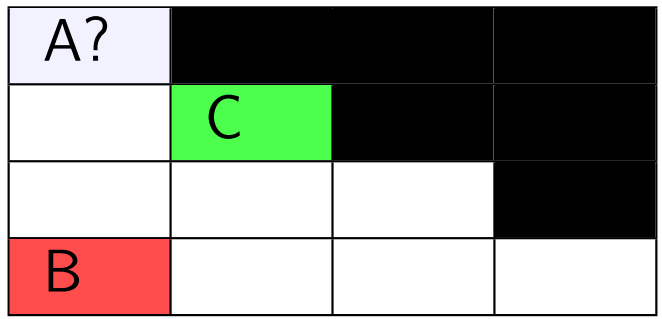
$w = aabb$

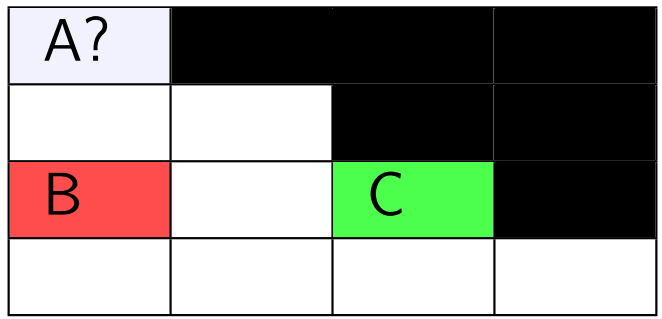
(0,3)			
(0,2)	(1,3)		
(0,1)	(1,2)	(2,3)	
(0,0)	(1,1)	(2,2)	(3,3)

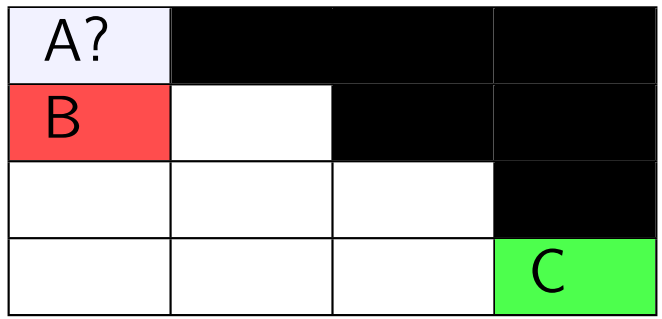
Fill in each successive diagonal by looking at rules of the form $A \to BC$, and some of the already filled in entries.

Illustration of which cells to pair up.

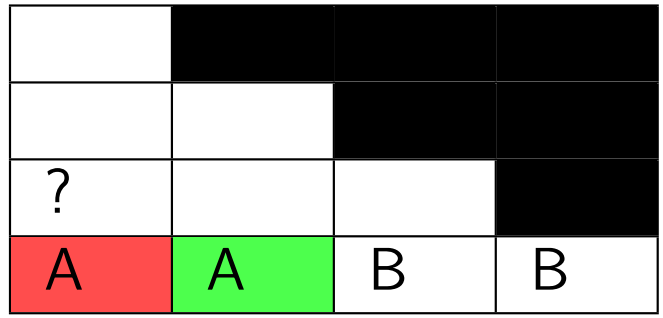
Let's take a rule $A \to BC$...


A?
C
B


A?
B
C


A?
B
C

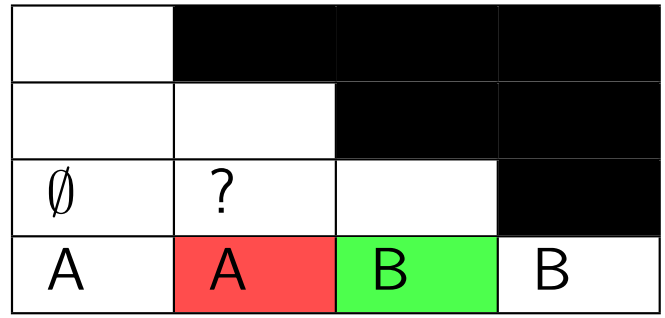
$$S \to AB \mid AX \mid \epsilon$$
$$T \to AB \mid AX$$
$$X \to TB$$
$$A \to a$$
$$B \to b$$



$w = aabb$

Put nothing in

$$S \to AB \mid AX \mid \epsilon$$
$$T \to AB \mid AX$$
$$X \to TB$$
$$A \to a$$
$$B \to b$$



$w = aabb$

Put both S and T in!

$$S \to AB \mid AX \mid \epsilon$$
$$T \to AB \mid AX$$
$$X \to TB$$
$$A \to a$$
$$B \to b$$

$\emptyset$	S,T	?	
A	A	B	B

$w = aabb$

Put nothing in

$$S \to AB \mid AX \mid \epsilon$$
$$T \to AB \mid AX$$
$$X \to TB$$
$$A \to a$$
$$B \to b$$

$\emptyset$	S,T	$\emptyset$	
A	A	B	B

$w = aabb$

$$S \to AB \mid AX \mid \epsilon$$
$$T \to AB \mid AX$$
$$X \to TB$$
$$A \to a$$
$$B \to b$$

?			
$\emptyset$	S,T	$\emptyset$	
A	A	B	B

$w = aabb$

Put nothing in

$$S \to AB \mid AX \mid \epsilon$$
$$T \to AB \mid AX$$
$$X \to TB$$
$$A \to a$$
$$B \to b$$

?			
$\emptyset$	S,T	$\emptyset$	
A	A	B	B

$w = aabb$

Put nothing in

$$S \to AB \mid AX \mid \epsilon$$
$$T \to AB \mid AX$$
$$X \to TB$$
$$A \to a$$
$$B \to b$$

$\emptyset$	?		
$\emptyset$	S,T	$\emptyset$	
A	A	B	B

$w = aabb$

Put nothing in

$$S \to AB \mid AX \mid \epsilon$$
$$T \to AB \mid AX$$
$$X \to TB$$
$$A \to a$$
$$B \to b$$

$\emptyset$	?		
$\emptyset$	S,T	$\emptyset$	
A	A	B	B

$w = aabb$

Put X in!

$$S \to AB \mid AX \mid \epsilon$$
$$T \to AB \mid AX$$
$$X \to TB$$
$$A \to a$$
$$B \to b$$

?			
$\emptyset$	X		
$\emptyset$	S,T	$\emptyset$	
A	A	B	B

$$w = aabb$$

$$S \to AB \mid AX \mid \epsilon$$
$$T \to AB \mid AX$$
$$X \to TB$$
$$A \to a$$
$$B \to b$$

S,T			
$\emptyset$	X		
$\emptyset$	S,T	$\emptyset$	
A	A	B	B

$w = aabb$

Put both S and T in!

$$S \to AB \mid AX \mid \epsilon$$
$$T \to AB \mid AX$$
$$X \to TB$$
$$A \to a$$
$$B \to b$$

S,T			
$\emptyset$	X		
$\emptyset$	S,T	$\emptyset$	
A	A	B	B

$w = aabb$

Put nothing in

$$S \to AB \mid AX \mid \epsilon$$
$$T \to AB \mid AX$$
$$X \to TB$$
$$A \to a$$
$$B \to b$$

S,T			
$\emptyset$	X		
$\emptyset$	S,T	$\emptyset$	
A	A	B	B

$w = aabb$

Put nothing in

$$S \to AB \mid AX \mid \epsilon$$
$$T \to AB \mid AX$$
$$X \to TB$$
$$A \to a$$
$$B \to b$$

S,T			
$\emptyset$	X		
$\emptyset$	S,T	$\emptyset$	
A	A	B	B

$w = aabb$

Done!

Good to know

What if I want to compute a derivation/parse-tree?

When computing the entries of the table, store the needed information.

Idea: if A is in *table*(i, j) store a rule $A \to BC$ and a number k — I like to call a "split" — that witnesses why A was put into *table*(i, j).

The 'split' k means we will find B in *table*(i, k) and C in *table*$(k + 1, j)$.

Good to know

- This is called the CYK algorithm (Cocke–Younger–Kasami). It is sometimes presented in the language of dynamic programming, an algorithmic technique you will learn in COMP3027.
- For fixed G and varying w, it runs in time $O(|w|^3)$.
- There is a subcubic algorithm (due to Leslie Valiant) for parsing CFGs in CNF based on the equally interesting fact that Matrix Multiplication can be done in subcubic time (due to Volker Strassen).
- If the input is large (e.g., a compiling a very large program), then $O(|w|^3)$ is too high. So, one often resorts to using restricted grammars for which there are linear-time algorithms (see "LR parsing").

Good to know

There is a machine-theoretic characterisation of context-free languages . . .

- Pushdown automaton = NFA + stack
- See Sipser Chapter 2.2

Where are we going?

Next, we study a very powerful model of computation that captures our intuitive idea of what a general algorithm can do. It is called the Turing Machine, and you can think of it like a finite automaton that has access to a specific data structure called a 'tape' which can be represented by a doubly-linked list.