COMP2022|2922

Models of Computation

Lesson 9a: C

Presente
Sasha Rubin
School of Computer Science

THE UNIVERSITY OF SYDNEY

– A contex-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.
– H

otherwise.

– This basic problem is solved by compilers and pa

Possible approaches...

1. Systematically search through all derivations (or all parse-trees) until you find one that derives w.
   - Try all i-step derivations for $i = 1, 2, 3 \ etc$
   - Problem: When to stop and declare "the string w cannot be derived from G"?
2. Us
   - Systematically compute, for every substr
     non-terminals of G derive w (if a
   - Then check if the start state S
     whole string w.
   - The resulting algorithm takes polynomial time in the worst case, i.e., is acceptably fast.
   - This is the approach we will take today! It is called the CYK algorithm

Input: a CFG G and string w.
Output: 1 if the string w is generated by G, and 0 otherwise.

Plan

1. Convert G into a grammar G' such that $L(G) = L(G')$ and

2. A e

Skeptic

Q. Why do we do the normal form?

A. In order to make the table small and easier to imple

E.g., the parse-trees of a grammar in CNF are binary trees!

Chomsky Normal Form

Definition

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

- $A \to BC$ (A, B, C are any variables, except that neither B
-
-

In the next slides, we will give a 5-step algorithm that tr every CFG into an equivalent one in Chomsky Nor

1. START: Eliminate the start symbol from the R
2. TERM: Eliminate rules with terminals, except for rules $A \to a$
3. BIN: Eliminate rules with more than two variables
4. DEL: Eliminate epsilon productions
5. UNIT: Eliminate unit rules

Chomsky Normal Form: algorithm

1. **Eliminate the start symbol from the RHS of all rules**
2. Eliminate rules with terminals, except for rules $A \to a$
3. Eliminate rules with more than two variables
4. Eliminate epsilon productions
5. Eli

Add t S_0 S_0 S

Chomsky Normal Form: algorithm

1. Eliminate the start symbol from the RHS of all rules
2. **Eliminate rules with terminals, except for rules** $A \to a$
3. Eliminate rules with more than two variables
4. Eliminate epsilon productions
5. Eli

- Replace every terminal a on the RHS the form $A \to a$) by the new variable
- For each such terminal a create the

Chomsky Normal Form: algorithm

1. Eliminate the start symbol from the RHS of all rules
2. Eliminate rules with terminals, except for rules $A \to a$
3. Eliminate rules with more than two variables
4. Eliminate epsilon productions
5. Eli

For ev), delete it and create new variables $A_1, A_2, \ldots, A$

$$A \to X_1 A_1$$
$$A_1 \to X_2 A_2$$
$$\vdots$$
$$A_{n-3} \to X_{n-2} A_{n-2}$$
$$A_{n-2} \to X_{n-1} X_n$$

Chomsky Normal Form: algorithm

1. Eliminate the start symbol from the RHS of all rules
2. Eliminate rules with terminals, except for rules $A \to a$
3. Eliminate rules with more than two variables
4. **Eliminate epsilon productions**
5. Eli

For every rule of the form $U \to \varepsilon$ (except $S_0 \quad \varepsilon$)

- Remove the rule.
- For each rule $A \to \alpha$ containing
  $A \to \alpha'$ where α' is α with one or mor
  the rule $A \to \epsilon$ if this rule has not already been removed.

Chomsky Normal Form: algorithm

1. Eliminate the start symbol from the RHS of all rules
2. Eliminate rules with terminals, except for rules $A \to a$
3. Eliminate rules with more than two variables
4. Eliminate epsilon productions
5. ---

For ea $A \to B$

- Remove the rule $A \to B$
- For each rule of the form $B \to \alpha$ (unless it was previously removed)

Chomsky Normal Form: example

$$S \to ASA \mid aB$$
$$A \quad B \quad S$$

Step 1 (

$$\mathbf{S_0} \to \mathbf{S}$$
$$S \to ASA \mid$$
$$A \to B \mid S$$
$$B \to b \mid \varepsilon$$

Chomsky Normal Form: example

$S_0 \to S$

$S \to ASA \mid aB$

$A \quad B \quad S$

Step 2 (

$A \to a$:

$S_0 \to S$

$S \to ASA \mid$

$A \to B \mid S$

$B \to b \mid \varepsilon$

$\mathbf{N_a} \to \mathbf{a}$

Chomsky Normal Form: example

$$S_0 \to S$$
$$S \to AS \mid N_a B$$
$$A \to B \mid S$$

Step 3 (

$$S_0 \to S$$
$$S \to \mathbf{AS_1} \mid$$
$$\mathbf{S_1} \to \mathbf{SA}$$
$$A \to B \mid S$$
$$B \to b \mid \varepsilon$$
$$N_a \to a$$

Chomsky Normal Form: example

$$S_0 \to S$$
$$S \to AS_1 \mid N_a B$$
$$S_1 \to SA$$

Step 4 (DEL): Eliminate epsilon production

$$S_0 \to S$$
$$S \to AS_1 \mid N_a \qquad \mathbf{a}$$
$$S_1 \to SA$$
$$A \to B \mid S \mid \varepsilon$$
$$B \to b$$
$$N_a \to a$$

Chomsky Normal Form: example

$$S_0 \to S$$

$$S \to AS_1 \mid N_aB \mid N_a$$

$$S_1 \to SA$$

Step 4 (DEL): Eliminate epsilon production

$$S_0 \to S$$

$$S \to AS_1 \mid N_aB \qquad a \qquad \mathbf{1}$$

$$S_1 \to SA \mid \mathbf{S}$$

$$A \to B \mid S$$

$$B \to b$$

$$N_a \to a$$

Chomsky Normal Form: example

$$S_0 \to S$$
$$S \to AS_1 \mid N_aB \mid N_a \mid S_1$$
$$S_1 \to SA \mid S$$

Step 5 (UNIT): Eliminate unit rules S

$$S_0 \to S$$
$$S \to AS_1 \mid N_aB \mid \mathbf{a} \mid \mathbf{SA}$$
$$S_1 \to SA \mid S$$
$$A \to \mathbf{b} \mid S$$
$$B \to b$$
$$N_a \to a$$

Chomsky Normal Form: example

$$S_0 \to S$$

$$S \to AS_1 \mid N_aB \mid a \mid SA$$

$$S_1 \to SA \mid S$$

Step 5 (UNIT): Eliminate unit rules S_0

$$S_0 \to AS_1 \mid N_aB \mid$$

$$S \to AS_1 \mid N_aB \mid$$

$$S_1 \to \mathbf{AS_1} \mid \mathbf{N_aB} \mid \mathbf{a} \mid SA$$

$$A \to b \mid \mathbf{AS_1} \mid \mathbf{N_aB} \mid \mathbf{a} \mid \mathbf{SA}$$

$$B \to b$$

$$N_a \to a$$

Chomsky Normal Form: example

All done.

$$A \to b \mid AS_1 \mid N_a$$

$$B \to b$$

$$N_a \to a$$

COMP2022|2922

Models of Computation

Lesson 9b: Membership problem for CFGs in C

Presente

Sasha Rubin
School of Computer Science

THE UNIVERSITY OF SYDNEY

Membership problem for CFG in CNF

Input: a CFG G that is in CNF, and string w.

Output: 1 if the string w is generated by G, and 0 otherwise.

Table-filling algorithm (aka Dynamic Programming)

- T

- The table records the solution to the subproble need to solve each subproblem once (aka me

- Main steps: define the subproblems, find the re sure you solve each subproblem once.
- You will see this again in `COMP3027:Algorithm Design`
- The algorithm we will see is known as the CYK algorithm (Cocke–Younger–Kasami).

What are the subproblems?

Idea

- A parse tree for a string w is built from a root, a left subtree, and a right subtree (remember that G is in CNF).
- The left (right) subtree is parse tree of a prefix (suffix) of w.
- So
- So

generate which substrings of w.

What are the entries in the table?

Given G in CNF, and a non-empty string $w = w_1 w_2 \cdots w_n$:

- Write $table(i, j)$ for the set of variables A that generate the substring $w_i w_{i+1} \cdots w_j$.
- The algorithm will compute $table(i, j)$ for all $1 \leq i < j \leq n$.
- O $n)$. If

Computing the table recursively

Compute $table(i, j)$ using the following recursive procedure:

1. If $i = j$ then $table(i, j)$ is the set of variables A such that $A \quad w_i$ is a rule of the grammar.

2 hich j

Q: Why is the recursion correct?

Computing the table recursively

Compute $table(i, j)$ using the following recursive procedure:

1. If $i = j$ then $table(i, j)$ is the set of variables A such that $A \quad w_i$ is a rule of the grammar.

2 hich i

Q: Why does this recursion stop?

- At each step, we call the procedure on "smaller" pr
- In what sense is $table(i, k)$ and $table(k + 1, j)$ smaller than $table(i, j)$? The size of the intervals $[i, k]$ and $[k + 1, j]$ is smaller than the size of the interval $[i, j]$.

We want to avoid computing table entries more than once.

- So, before making a recursive call just check if the value has already been computed. If yes, use that value and don't recurse. If not, recurse.

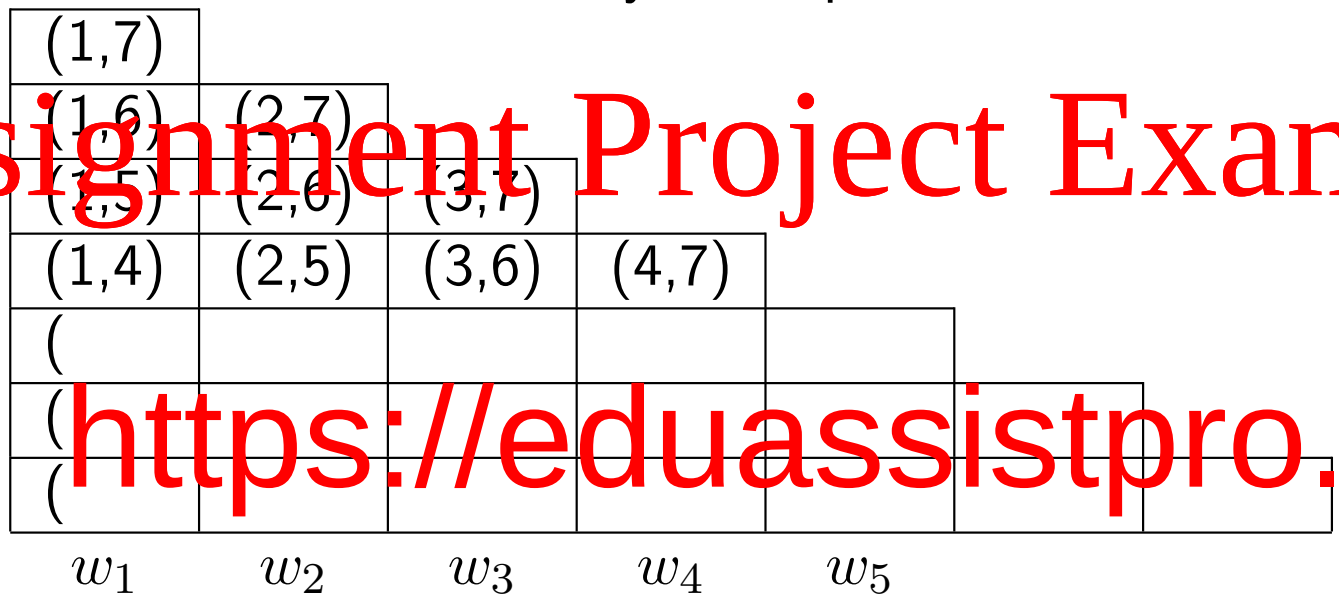
Can we write this with a bunch of loops?

Yes, but it is harder to read. See Sipser (edition 3) Theorem 7.16.

(pseudocode from "Introduction to the theory of computation" by Michael Sipser)

Can we fill the table by hand?

Yes, but this is best done by a computer!



Q: Where do I put the entry $table(i, j)$

- horizontal co-ordinate = starting position of s
- vertical co-ordinate = length of substring =

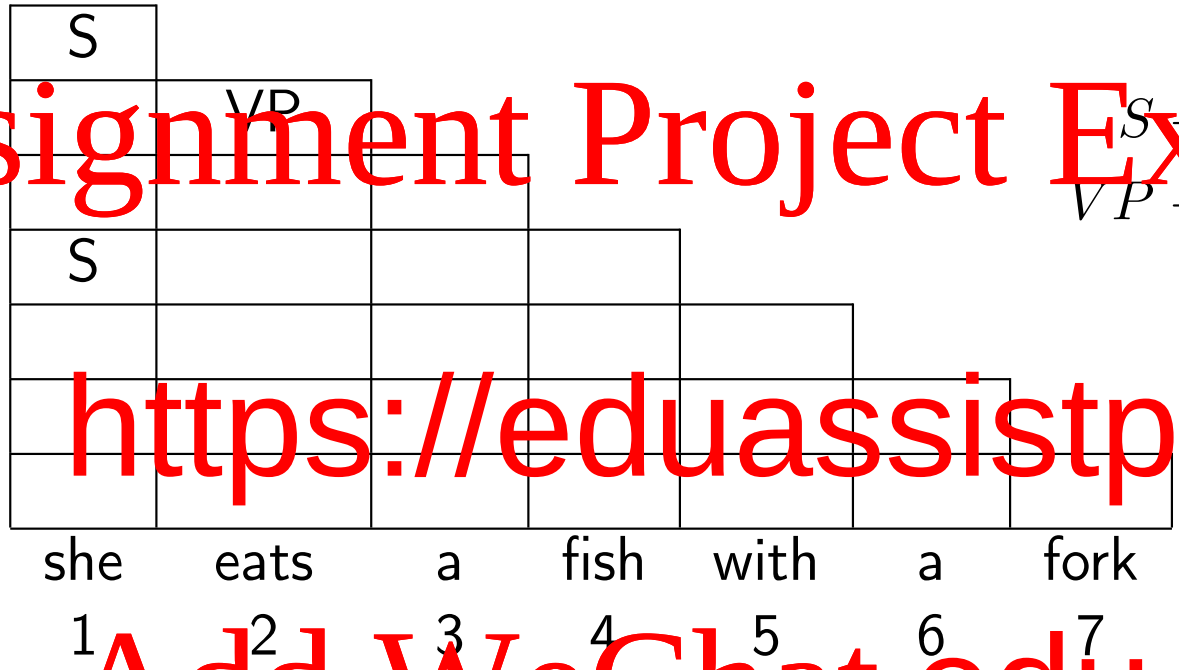
Q: What entries are needed to compute $table(i, j)$?

- You have to look at the pairs $table(i, k), table(k + 1, j)$ for $k = i, \ldots, j - 1$; it's the right-angled triangle below $table(i, j)$.

Q: In what order are the entries computed?

- Row by row, bottom to top, left to right

Example



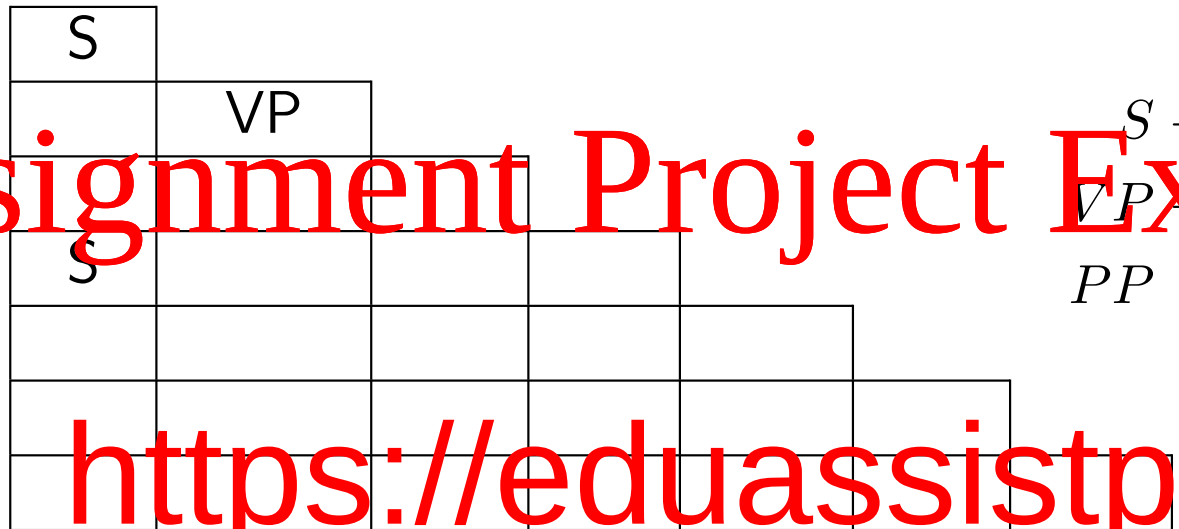
$S \rightarrow NP\ VP$

$VP \rightarrow VP\ PP \mid V\ NP$

Q: In what order are the entries computed?

- To compute an entry, you need the entries in the "right-angled triangle below it".
- Row by row, bottom to top, left to right

Example



$S \rightarrow NP\ VP$

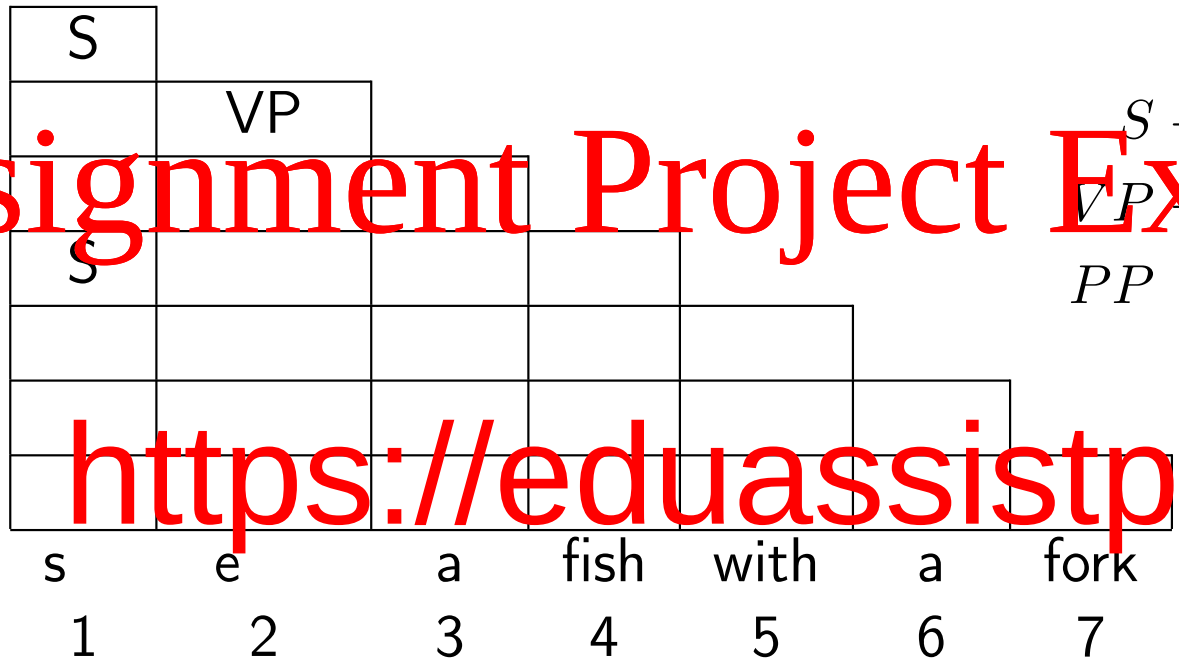
$VP \rightarrow VP\ PP \mid V\ NP$

$PP \quad P\ NP$

$VP \in table(2,4)$ because

- the string from position 2 to 4 is "e
- which can be split into "eats" from position 2 to 2,
- and "a fish" from position 3 to 4, and
- and $VP \rightarrow V\ NP$ is a rule, $V \in table(2,2)$, and $NP \in table(3,4)$.

Example



$S \rightarrow NP\ VP$

$VP \rightarrow VP\ PP \mid V\ NP$

$PP \quad P\ NP$

$VP \in table(2,7)$ because

- the string from position 2 to 7 is "e
- which can be split into "eats a fish" from position 2 to 4,
- and "with a fork" from position 5 to 7, and
- and $VP \rightarrow VP\ PP$ is a rule, $VP \in table(2,4)$, and $PP \in table(5,7)$.

How efficient is this algorithm?

Time complexity

- There are $O(n^2)$ entries in the table,
- and each entry requires $O(n|G|)$ work to compute, since one must check each rule and check at most n splits,
- So t
- H

  rules, variables and terminals.

Asides

- For fixed G and varying w, the time is
- If the input is large (e.g., a compiling a very large program), then this complexity is too high. So, in this case, one uses restricted grammars for which there are faster algorithms, see `COMP3109:Programming Languages and Paradigms`

What if I want to compute a derivation?

You can adjust the algorithm to store more information in order to produce a derivation (or parse tree)

- Store in $table(i, j)$ a rule A BC and splitting point k

  $..w_j$.

- Y

- St

  then repeat the following:

  - if (A, i, i) is the top element of the stack t
    $A \to w_i$ and pop the stack.
  - if (A, i, j) is the top-element of the sta
    $A \to BC$, pop the stack, and push the element (B, i, k) followed by $(C, k + 1, j)$ onto the stack.

Good to know

- There is a machine-theoretic characterisation of context-free languages (pushdown automaton = NFA + stack).
  - Come to COMP2922 to learn more! or see Sipser Chapter 2.2
- Not every language is context-free. E.g., $\{ww : w \in \{0,1\}^*\}$ is not context-free.

Where are we going?

Next week we start learning about an even more powerful model of com

the T

This is the most powerful model of computation th

and is a model of a general purpose computer.