



Aalto University
School of Science

CS-C2160 Theory of Computation

Lecture 7: The CYK Parsing Algorithm and Chomsky Normal Form, Pushdown Automata, Limitations of Context-Free Languages

Pekka Orponen
Aalto University
Department of Computer Science

Topics:

- Parsing context-free grammars
- The Chomsky normal form for CFGs
- The CYK parsing algorithm
- Pushdown automata
- Pushdown automata and context-free languages
- A pumping lemma for context-free languages

Material:

- In Finnish: Sections 3.6–3.8 in the Finnish lecture notes
- In English: In the Sipser book as follows:
 - ▶ Chomsky normal form: Section 2.1
 - ▶ CYK algorithm: Theorem 7.16
 - ▶ Pushdown automata: Section 2.2
 - ▶ Limitations of context-free languages: Section 2.3

Recap: Context-free grammars

Example:

A (simplified) grammar for arithmetic expressions with function calls in a C-like programming language:

$$E \rightarrow T \mid E + T$$

$$T \rightarrow F \mid T * F$$

$$F \rightarrow a \mid (E) \mid f(L)$$

$$L \rightarrow \varepsilon \mid L'$$

$$L' \rightarrow E \mid E, L'$$


Deriving the string $f(a+a)*a$ in the grammar:

$$\begin{aligned} \underline{E} &\Rightarrow \underline{T} &\Rightarrow \underline{T} * F &\Rightarrow \underline{F} * F \\ &\Rightarrow f(\underline{L}) * F &\Rightarrow f(\underline{L}') * F &\Rightarrow f(\underline{E}) * F \\ &\Rightarrow f(\underline{E} + T) * F &\Rightarrow f(\underline{T} + T) * F &\Rightarrow f(\underline{E} + T) * F \\ &\Rightarrow f(a + \underline{T}) * F &\Rightarrow f(a + \underline{F}) * F &\Rightarrow f(a + a) * \underline{F} \\ &\Rightarrow f(a + a) * a. \end{aligned}$$

The CYK Parsing Algorithm and Chomsky Normal Form

7.1 Parsing context-free grammars

- The parsing problem: given a context-free grammar G and a string x , is $x \in \mathcal{L}(G)$, i.e. can one derive x using the rules of G ?
- Previous lecture: recursive-descent parsing, LL(1) grammars.
- Recursive descent parsing is an effective parsing method for LL(1) grammars: strings of length n can be parsed in time $O(n)$.
- However, LL(1) grammars are a rather restricted class of context-free grammars, and the parsing problem for context-free grammars in general is not as easy.
- *In principle* one can always use recursive-descent parsing with backtracking, but *in practice* this is not feasible due to the very large number possible derivations. (Typically $O(c^n)$ for some $c \geq 2$.)

- The *Cocke-Younger-Kasami (CYK) algorithm* is a “dynamic programming” algorithm for parsing context-free grammars.
- It works in time $O(n^3)$, where n is the length of the string to be parsed. (Here the size of the grammar is assumed to be a constant and thus ignored.)
- In order to use the CYK algorithm, it is helpful to make some transformations to the grammar at hand.
 We first transform the grammar into *Chomsky normal form*, where we only need to consider productions of a few simple basic types.

7.2 The Chomsky normal form for CFGs

- We say that a variable A is *nullable* (in grammar G) if $A \Rightarrow_G^* \epsilon$.

Definition 7.1

A context-free grammar $G = (V, \Sigma, P, S)$ is in *Chomsky normal form* if

1. none of the variables, except possibly the start variable, are nullable,
2. except for the possible production $S \rightarrow \epsilon$, all the productions are of form

$$A \rightarrow BC \quad \text{or} \quad A \rightarrow a$$

where A, B, C are variables and a is a terminal symbol, and

3. the start variable S does not occur on the right-hand side of any production.

1. Removing ϵ -productions

- Let $G = (V, \Sigma, P, S)$ be a context-free grammar.
- Recall: a variable $A \in V - \Sigma$ is *nullable* if $A \Rightarrow_G^* \epsilon$.

Example:

In the grammar

$$\begin{aligned} S &\rightarrow A \mid B \\ A &\rightarrow aBa \mid \epsilon \\ B &\rightarrow bAb \mid \epsilon \end{aligned}$$

all the variables S, A, B are nullable.

Lemma 7.1

For any context-free grammar G , we can construct an equivalent grammar G' (that is, G' generates the same language as G) in which none of the variables, except possibly the start variable, are nullable.

Proof

Let $G = (V, \Sigma, P, S)$. First, determine all the nullable variables in G :

1. Start by setting

$$\text{NULL} := \{A \in V - \Sigma \mid A \rightarrow \varepsilon \text{ is a production in } G\}$$

2. and then expand the NULL set as long as possible with the operation

$$\begin{aligned} \text{NULL} &:= \text{NULL} \cup \\ &\quad \{A \in V - \Sigma \mid A \rightarrow B_1 \dots B_k \text{ is a production in } G \text{ and} \\ &\quad B_i \in \text{NULL for all } i = 1, \dots, k\}. \end{aligned}$$

After this, replace each production $A \rightarrow X_1 \dots X_k$ in G with the set of all productions of the form

$$A \rightarrow \alpha_1 \dots \alpha_k, \text{ where}$$
$$\alpha_i = \begin{cases} X_i & \text{if } X_i \notin \text{NULL}, \\ X_i \text{ or } \varepsilon & \text{if } X_i \in \text{NULL}. \end{cases}$$

Finally, remove all productions of form $A \rightarrow \varepsilon$. If the production $S \rightarrow \varepsilon$ is removed, add a new start variable S' and the productions $S' \rightarrow S$ and $S' \rightarrow \varepsilon$.

Example:

Let us remove the ε -productions from the grammar

$$\begin{aligned} S &\rightarrow A \mid B \\ A &\rightarrow aBa \mid \varepsilon \\ B &\rightarrow bAb \mid \varepsilon \end{aligned}$$

Now $\text{NULL} = \{A, B, S\}$ and we get

$$\begin{aligned} S &\rightarrow A \mid B \mid \varepsilon \\ A &\rightarrow aBa \mid aa \mid \varepsilon \\ B &\rightarrow bAb \mid bb \mid \varepsilon \end{aligned}$$

and finally

$$\begin{aligned} S' &\rightarrow S \mid \varepsilon \\ S &\rightarrow A \mid B \\ A &\rightarrow aBa \mid aa \\ B &\rightarrow bAb \mid bb \end{aligned}$$

2. Removing unit productions

- A production of form $A \rightarrow B$, where A and B are variables, is called a *unit production*.

Lemma 7.2

For any context-free grammar G , we can construct an equivalent context-free grammar G' that does not contain any unit productions.

Proof

Let $G = (V, \Sigma, P, S)$. First compute the “unit successors” set $F(A)$ for each variable A as follows:

1. First, for each variable A , set

$$F(A) := \{B \mid A \rightarrow B \text{ is a production in } G \text{ and } B \text{ is a variable}\}$$

2. and then expand the F -sets as long as possible with the operation

$$F(A) := F(A) \cup \bigcup \{F(B) \mid A \rightarrow B \text{ is a production in } G\}.$$

After this:

1. remove all the unit productions from G , and
2. for every variable A , add all possible productions of form $A \rightarrow \omega$, where $B \rightarrow \omega$ is a non-unit production in G for some $B \in F(A)$.

Example:

Let us remove the unit productions from our earlier grammar

$$S' \rightarrow S \mid \varepsilon$$

$$S \rightarrow A \mid B$$

$$A \rightarrow aBa \mid aa$$

$$B \rightarrow bAb \mid bb$$

The unit successor sets for the variables are: $F(S') = \{S, A, B\}$, $F(S) = \{A, B\}$, $F(A) = F(B) = \emptyset$.

By replacing the unit productions as described, we get the grammar

$$S' \rightarrow aBa \mid aa \mid bAb \mid bb \mid \varepsilon$$

$$S \rightarrow aBa \mid aa \mid bAb \mid bb$$

$$A \rightarrow aBa \mid aa$$

$$B \rightarrow bAb \mid bb$$

(Note that the variable S is now redundant, i.e. it cannot occur in any derivation. One can remove all redundant variables from a grammar with a similar algorithm. Details of this are left as an exercise.)

Theorem 7.3

For any context-free grammar G , we can construct an equivalent grammar G' that is in Chomsky normal form.

Proof

Let $G = (V, \Sigma, P, S)$ be a CFG.

1. If the start variable S occurs on the right-hand side of any production, introduce a new start variable S' and add the production $S' \rightarrow S$.
2. Remove all the ε - and unit productions by using the constructions of Lemmas 3.1 and 3.2.

After this, each production is of one of the following forms:

- $A \rightarrow a$, where a is a terminal,
- $A \rightarrow X_1 \dots X_k$, where $k \geq 2$, or
- $S' \rightarrow \varepsilon$.

To conclude the transformation, do the following:

1. For each terminal a , add a new variable C_a and the production $C_a \rightarrow a$.
2. In each production of form $A \rightarrow X_1 \dots X_k$, $k \geq 2$, first replace each terminal a with the corresponding variable C_a , and then the whole production with the production set

$$\begin{aligned} A &\rightarrow X_1 A_1 \\ A_1 &\rightarrow X_2 A_2 \\ &\vdots \\ A_{k-2} &\rightarrow X_{k-1} X_k, \end{aligned}$$

where A_1, \dots, A_{k-2} are again new variables.

Example:

Consider the grammar:

$$\begin{aligned} S &\rightarrow aBCd \\ S &\rightarrow bbb \\ B &\rightarrow b \\ C &\rightarrow c \end{aligned}$$

A Chomsky normal form grammar obtained with the above construction:

$$\begin{aligned} S &\rightarrow C_a S_1^1 \\ S_1^1 &\rightarrow B S_2^1 \\ S_2^1 &\rightarrow C C_d \\ S &\rightarrow C_b S_1^2 \\ S_1^2 &\rightarrow C_b C_b \\ B &\rightarrow b \\ C &\rightarrow c \\ C_a &\rightarrow a \\ C_b &\rightarrow b \\ C_c &\rightarrow c \\ C_d &\rightarrow d \end{aligned}$$

7.3 The CYK parsing algorithm

- Consider a CFG $G = (V, \Sigma, P, S)$.
- By Thm 7.3, we may assume that G is in Chomsky normal form.
- An intuition behind the algorithm: due to the Chomsky normal form conditions:

1. Only the start variable S can derive the empty string ϵ , and this can only happen if G has the production $S \rightarrow \epsilon$.
2. The only variables X that can derive a string a of length one are the ones that have a production of form $X \rightarrow a$.
3. The other productions are of form $X \rightarrow AB$. These derive strings of form $a_1 \dots a_k$, where for some $1 \leq l < k$: variable A derives substring $a_1 \dots a_l$ and variable B derives substring $a_{l+1} \dots a_k$.

Based on these observations, we can devise a tabulating (“dynamic programming”) algorithm that iteratively considers longer and longer substrings of a target string, using the information already available for shorter substrings.

- More specifically, the problem of deciding whether a given string x belongs to the language $\mathcal{L}(G)$ can be solved as follows:

- ▶ If $x = \epsilon$, then $x \in \mathcal{L}(G)$ if and only if $S \rightarrow \epsilon$ is a production in G .
- ▶ Otherwise, $x = a_1 \dots a_n$ and we study how substrings of x can be derived.

Let $N_{i,k}$ denote the set of variables A from which one can derive the substring of x that starts at index i and has length k :

$$N_{i,k} = \{A \mid A \xRightarrow{*}_G a_i \dots a_{i+k-1}\}, \\ 1 \leq i \leq i+k-1 \leq n.$$

Clearly $x \in \mathcal{L}(G)$ if and only if $S \in N_{1,n}$.

The sets $N_{i,k}$ can be computed iteratively from shorter substrings to longer ones as described on the next slide.

- Computing the sets $N_{i,k}$:

- ▶ First, for each $i = 1, \dots, n$, assign

$$N_{i,1} := \{A \mid A \rightarrow a_i \text{ is a production in } G\}.$$

- ▶ Next, for each $k = 2, \dots, n$, and for each $i = 1, \dots, n - k + 1$, compute

$$N_{i,k} := \bigcup_{j=1}^{k-1} \{A \mid A \rightarrow BC \text{ is a production in } G \text{ and} \\ B \in N_{i,j} \text{ and } C \in N_{i+j,k-j}\}.$$

Example:

Consider the grammar G in Chomsky normal form:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

Executing the CYK algorithm on target string $x = baaba$:

$N_{i,k}$	$i \rightarrow$				
	1 : b	2 : a	3 : a	4 : b	5 : a
1	B				
2					
$k \downarrow$ 3					
4					
5					

$N_{1,1} = \{B\}$ as $B \rightarrow b$ is the only production that can derive the substring b .

Example:

Consider the grammar G in Chomsky normal form:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

Executing the CYK algorithm on target string $x = baaba$:

$N_{i,k}$	$i \rightarrow$				
	1 : b	2 : a	3 : a	4 : b	5 : a
1	B	A, C			
2					
$k \downarrow$ 3					
4					
5					

$N_{2,1} = \{A, C\}$ as $A \rightarrow a$ and $C \rightarrow a$ are the only productions that can derive the substring a .

Example:

Consider the grammar G in Chomsky normal form:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

Executing the CYK algorithm on target string $x = baaba$:

$N_{i,k}$	$i \rightarrow$				
	1 : b	2 : a	3 : a	4 : b	5 : a
1	B	A, C	A, C	B	A, C
2					
$k \downarrow$ 3					
4					
5					

Similarly for all $N_{i,1}$, where $1 \leq i \leq 5$

Example:

Consider the grammar G in Chomsky normal form:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

Executing the CYK algorithm on target string $x = baaba$:

$N_{i,k}$	$i \rightarrow$				
	1 : b	2 : a	3 : a	4 : b	5 : a
1	B	A, C	A, C	B	A, C
2	S, A				
$k \downarrow$ 3					
4					
5					

$N_{1,2} = \{S, A\}$ as $S \rightarrow BC$ and $A \rightarrow BA$ are the productions whose right-hand sides are in the set $N_{1,1} \times N_{2,1} = \{BA, BC\}$.

Example:

Consider the grammar G in Chomsky normal form:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

Executing the CYK algorithm on target string $x = baaba$:

	$N_{i,k}$	$i \rightarrow$				
		1 : b	2 : a	3 : a	4 : b	5 : a
	1	B	A, C	A, C	B	A, C
	2	S, A	B			
$k \downarrow$	3					
	4					
	5					

$N_{2,2} = \{B\}$ as $B \rightarrow CC$ is the only production whose right-hand side is in the set $N_{2,1} \times N_{3,1} = \{AA, AC, CA, CC\}$.

Example:

Consider the grammar G in Chomsky normal form:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

Executing the CYK algorithm on target string $x = baaba$:

	$N_{i,k}$	$i \rightarrow$				
		1 : b	2 : a	3 : a	4 : b	5 : a
	1	B	A, C	A, C	B	A, C
	2	S, A	B	S, C	S, A	
$k \downarrow$	3					
	4					
	5					

Similarly for all $N_{i,2}$, where $1 \leq i \leq 4$.

Example:

Consider the grammar G in Chomsky normal form:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

Executing the CYK algorithm on target string $x = baaba$:

	$N_{i,k}$	$i \rightarrow$				
		1 : b	2 : a	3 : a	4 : b	5 : a
	1	B	A, C	A, C	B	A, C
	2	S, A	B	S, C	S, A	
$k \downarrow$	3	\emptyset				
	4					
	5					

$N_{1,3} = \emptyset$ as there are no productions with right-hand sides in the set $(N_{1,1} \times N_{2,2}) \cup (N_{1,2} \times N_{3,1}) = \{BB, SA, SC, AA, AC\}$.

Example:

Consider the grammar G in Chomsky normal form:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

Executing the CYK algorithm on target string $x = baaba$:

	$N_{i,k}$	$i \rightarrow$				
		1 : b	2 : a	3 : a	4 : b	5 : a
	1	B	A, C	A, C	B	A, C
	2	S, A	B	S, C	S, A	
$k \downarrow$	3	\emptyset	B			
	4					
	5					

$N_{2,3} = \{B\}$ as $B \rightarrow CC$ is the only production with right-hand side in the set $(N_{2,1} \times N_{3,2}) \cup (N_{2,2} \times N_{4,1}) = \{AS, AC, CS, CC, BB\}$.

Example:

Consider the grammar G in Chomsky normal form:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

Executing the CYK algorithm on target string $x = baaba$:

		$i \rightarrow$				
	$N_{i,k}$	1 : b	2 : a	3 : a	4 : b	5 : a
	1	B	A, C	A, C	B	A, C
	2	S, A	B	S, C	S, A	
$k \downarrow$	3	\emptyset	B	B		
	4					
	5					

Similarly for all $N_{i,3}$, where $1 \leq i \leq 3$.

Example:

Consider the grammar G in Chomsky normal form:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

Executing the CYK algorithm on target string $x = baaba$:

		$i \rightarrow$				
	$N_{i,k}$	1 : b	2 : a	3 : a	4 : b	5 : a
	1	B	A, C	A, C	B	A, C
	2	S, A	B	S, C	S, A	
$k \downarrow$	3	\emptyset	B	B		
	4	\emptyset				
	5					

$N_{1,4} = \emptyset$ as there are no productions with right-hand sides in the set $(N_{1,1} \times N_{2,3}) \cup (N_{1,2} \times N_{3,2}) \cup (N_{1,3} \times N_{4,1}) = \{BB, SS, SC, AS, AC\}$.

Example:

Consider the grammar G in Chomsky normal form:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

Executing the CYK algorithm on target string $x = baaba$:

		$i \rightarrow$				
	$N_{i,k}$	1 : b	2 : a	3 : a	4 : b	5 : a
	1	B	A, C	A, C	B	A, C
	2	S, A	B	S, C	S, A	
$k \downarrow$	3	\emptyset	B	B		
	4	\emptyset	S, A, C			
	5	S, A, C				

And so on...

Example:

Consider the grammar G in Chomsky normal form:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

Executing the CYK algorithm on target string $x = baaba$:

		$i \rightarrow$				
	$N_{i,k}$	1 : b	2 : a	3 : a	4 : b	5 : a
	1	B	A, C	A, C	B	A, C
	2	S, A	B	S, C	S, A	
$k \downarrow$	3	\emptyset	B	B		
	4	\emptyset	S, A, C			
	5	S, A, C				

As the start variable S belongs to the set $N_{1,5}$, we deduce that x belongs to the language $\mathcal{L}(G)$.

Example:

Consider the grammar G in Chomsky normal form:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

Executing the CYK algorithm on target string $x = baaba$:

	$N_{i,k}$	$i \rightarrow$				
		1 : b	2 : a	3 : a	4 : b	5 : a
	1	B	A, C	A, C	B	A, C
	2	S, A	B	S, C	S, A	
$k \downarrow$	3	\emptyset	B	B		
	4	\emptyset	S, A, C			
	5	S, A, C				

We can construct a leftmost derivation (or a parse tree) by remembering why a variable belongs to an $N_{i,k}$ set:

Example:

Consider the grammar G in Chomsky normal form:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

Executing the CYK algorithm on target string $x = baaba$:

	$N_{i,k}$	$i \rightarrow$				
		1 : b	2 : a	3 : a	4 : b	5 : a
	1	B	A, C	A, C	B	A, C
	2	S, A	B	S, C	S, A	
$k \downarrow$	3	\emptyset	B	B		
	4	\emptyset	S, A, C			
	5	S, A, C				

$S \in N_{1,5}$ as $S \rightarrow BC$ and $B \in N_{1,1}$ and $C \in N_{2,4}$
 $\underline{S} \Rightarrow BC$

Example:

Consider the grammar G in Chomsky normal form:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

Executing the CYK algorithm on target string $x = baaba$:

	$N_{i,k}$	$i \rightarrow$				
		1 : b	2 : a	3 : a	4 : b	5 : a
	1	B	A, C	A, C	B	A, C
	2	S, A	B	S, C	S, A	
$k \downarrow$	3	\emptyset	B	B		
	4	\emptyset	S, A, C			
	5	S, A, C				

$B \in N_{1,1}$ as $B \rightarrow b$
 $S \Rightarrow \underline{BC} \Rightarrow bC$

Example:

Consider the grammar G in Chomsky normal form:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

Executing the CYK algorithm on target string $x = baaba$:

	$N_{i,k}$	$i \rightarrow$				
		1 : b	2 : a	3 : a	4 : b	5 : a
	1	B	A, C	A, C	B	A, C
	2	S, A	B	S, C	S, A	
$k \downarrow$	3	\emptyset	B	B		
	4	\emptyset	S, A, C			
	5	S, A, C				

$C \in N_{2,4}$ as $C \rightarrow AB$ and $A \in N_{2,1}$ and $B \in N_{3,3}$
 $S \Rightarrow BC \Rightarrow b\underline{C} \Rightarrow bAB$

Example:

Consider the grammar G in
Chomsky normal form:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

Executing the CYK algorithm on target string $x = baaba$:

		$i \rightarrow$				
	$N_{i,k}$	1 : b	2 : a	3 : a	4 : b	5 : a
	1	B	A, C	A, C	B	A, C
	2	S, A	B	S, C	S, A	
$k \downarrow$	3	\emptyset	B	B		
	4	\emptyset	S, A, C			
	5	S, A, C				

$A \in N_{2,1}$ as $A \rightarrow a$

$S \Rightarrow BC \Rightarrow bC \Rightarrow bAB \Rightarrow baB$

Example:

Consider the grammar G in
Chomsky normal form:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

Executing the CYK algorithm on target string $x = baaba$:

		$i \rightarrow$				
	$N_{i,k}$	1 : b	2 : a	3 : a	4 : b	5 : a
	1	B	A, C	A, C	B	A, C
	2	S, A	B	S, C	S, A	
$k \downarrow$	3	\emptyset	B	B		
	4	\emptyset	S, A, C			
	5	S, A, C				

$B \in N_{3,3}$ as $B \rightarrow CC$ and $C \in N_{3,2}$ and $C \in N_{5,1}$

$S \Rightarrow BC \Rightarrow bC \Rightarrow bAB \Rightarrow baB \Rightarrow baCC$

Example:

Consider the grammar G in
Chomsky normal form:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

Executing the CYK algorithm on target string $x = baaba$:

		$i \rightarrow$				
	$N_{i,k}$	1 : b	2 : a	3 : a	4 : b	5 : a
	1	B	A, C	A, C	B	A, C
	2	S, A	B	S, C	S, A	
$k \downarrow$	3	\emptyset	B	B		
	4	\emptyset	S, A, C			
	5	S, A, C				

$C \in N_{3,2}$ as $C \rightarrow AB$ and $A \in N_{3,1}$ and $B \in N_{4,1}$

$S \Rightarrow BC \Rightarrow bC \Rightarrow bAB \Rightarrow baB \Rightarrow baCC \Rightarrow baABC$

Example:

Consider the grammar G in
Chomsky normal form:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

Executing the CYK algorithm on target string $x = baaba$:

		$i \rightarrow$				
	$N_{i,k}$	1 : b	2 : a	3 : a	4 : b	5 : a
	1	B	A, C	A, C	B	A, C
	2	S, A	B	S, C	S, A	
$k \downarrow$	3	\emptyset	B	B		
	4	\emptyset	S, A, C			
	5	S, A, C				

$A \in N_{3,1}$ as $A \rightarrow a$

$S \Rightarrow BC \Rightarrow bC \Rightarrow bAB \Rightarrow baB \Rightarrow baCC \Rightarrow baABC \Rightarrow baaBC$

Example:

Consider the grammar G in Chomsky normal form:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

Executing the CYK algorithm on target string $x = baaba$:

$N_{i,k}$	$i \rightarrow$				
	$1 : b$	$2 : a$	$3 : a$	$4 : b$	$5 : a$
1	B	A, C	A, C	B	A, C
2	S, A	B	S, C	S, A	
$k \downarrow$ 3	\emptyset	B	B		
4	\emptyset	S, A, C			
5	S, A, C				

$B \in N_{4,1}$ as $B \rightarrow b$

$S \Rightarrow BC \Rightarrow bC \Rightarrow bAB \Rightarrow baB \Rightarrow baCC \Rightarrow baABC \Rightarrow baaBC \Rightarrow baabC$

Example:

Consider the grammar G in Chomsky normal form:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

Executing the CYK algorithm on target string $x = baaba$:

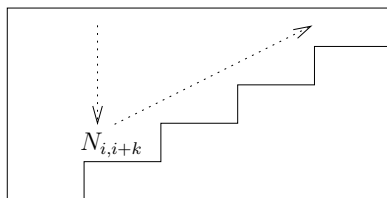
$N_{i,k}$	$i \rightarrow$				
	$1 : b$	$2 : a$	$3 : a$	$4 : b$	$5 : a$
1	B	A, C	A, C	B	A, C
2	S, A	B	S, C	S, A	
$k \downarrow$ 3	\emptyset	B	B		
4	\emptyset	S, A, C			
5	S, A, C				

$C \in N_{5,1}$ as $C \rightarrow a$

$S \Rightarrow BC \Rightarrow bC \Rightarrow bAB \Rightarrow baB \Rightarrow baCC \Rightarrow baABC \Rightarrow baaBC \Rightarrow baabC \Rightarrow baaba$

In general, when computing the a set $N_{i,k}$ in the CYK algorithm, one moves synchronously

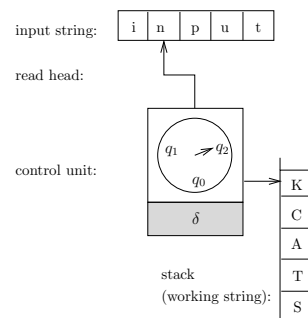
- in the column $N_{i,j}$ towards the set $N_{i,k}$ and
- in the diagonal $N_{i+j,k-j}$ away from the set $N_{i,k}$



Pushdown Automata

7.4 Pushdown automata

- The automata class corresponding to context-free grammars turns out to be (nondeterministic) *pushdown automata*:



- A pushdown automaton is like a finite automaton but it is extended with a *stack* of unlimited size.
- The use of stack is rather limited: the automaton can only read, write, add and delete symbols on the top of the stack (that is, it has no random-access to the symbols in the stack).

Definition 7.2

A *pushdown automaton* is a tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, F),$$

where

- Q is the finite set of *states*,
- Σ is the finite *input alphabet*,
- Γ is the finite *stack alphabet*,
- $\delta: Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q \times (\Gamma \cup \{\epsilon\}))$ is the (set-valued) *transition function*,
- $q_0 \in Q$ is the *start state*, and
- $F \subseteq Q$ is the set of *accept states*.

- The interpretation for a value

$$\delta(q, \sigma, \gamma) = \{(q_1, \gamma_1), \dots, (q_k, \gamma_k)\}$$

of the transition function is that, when in the state q and reading input symbol σ and stack symbol γ , the automaton can move to any one of the states q_1, \dots, q_k and replace the topmost symbol on the stack with the corresponding symbol $\gamma_1, \dots, \gamma_k$.

- Note that in pushdown automata, nondeterminism is allowed already in the basic definition!
- If $\sigma = \epsilon$, the automaton can take the transition without reading the next input symbol.
- If $\gamma = \epsilon$, the automaton does not read (and remove) the topmost symbol on the stack, but the new symbol is added on the stack on top of the previous one (the “push” operation).
- If the stack symbol read is $\gamma \neq \epsilon$ and the stack symbol to be written is $\gamma_i = \epsilon$, the topmost symbol of the stack is removed (the “pop” operation).

- A *configuration* of the automaton is a triple $(q, w, \alpha) \in Q \times \Sigma^* \times \Gamma^*$.
- In particular, the *start configuration* of the automaton on input x is the triple (q_0, x, ϵ) .
- Intuition: in configuration (q, w, α) the automaton is in state q , the input string that has not yet been processed is w , and the stack contains (from top to bottom) the symbols in α .
- A configuration (q, w, α) *leads in one step* to a configuration (q', w', α') , denoted as

$$(q, w, \alpha) \xrightarrow{M} (q', w', \alpha'),$$

if one can write $w = \sigma w'$, $\alpha = \gamma \beta$, $\alpha' = \gamma' \beta$ ($|\sigma|, |\gamma|, |\gamma'| \leq 1$), so that

$$(q', \gamma') \in \delta(q, \sigma, \gamma).$$

- A configuration (q, w, α) *leads* to a configuration (q', w', α') , denoted as

$$(q, w, \alpha) \vdash_M^* (q', w', \alpha'),$$

if there is a sequence $(q_0, w_0, \alpha_0), (q_1, w_1, \alpha_1), \dots, (q_n, w_n, \alpha_n)$, $n \geq 0$, of configurations such that

$$(q, w, \alpha) = (q_0, w_0, \alpha_0) \vdash_M (q_1, w_1, \alpha_1) \vdash_M \dots \vdash_M (q_n, w_n, \alpha_n) = (q', w', \alpha').$$

- A pushdown automaton M *accepts* a string $x \in \Sigma^*$ if

$$(q_0, x, \varepsilon) \vdash_M^* (q_f, \varepsilon, \alpha) \quad \text{for some } q_f \in F \text{ and } \alpha \in \Gamma^*,$$

meaning that it *can be* in an accept state when the whole input has been processed; otherwise, M *rejects* x .

- The language *recognised* by the automaton M is

$$\mathcal{L}(M) = \{x \in \Sigma^* \mid (q_0, x, \varepsilon) \vdash_M^* (q_f, \varepsilon, \alpha) \text{ for some } q_f \in F \text{ and } \alpha \in \Gamma^*\}.$$

Example:

A pushdown automaton for the language $\{a^k b^k \mid k \geq 0\}$:

$$M = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{A, \underline{A}\}, \delta, q_0, \{q_0, q_3\}),$$

where

$$\delta(q_0, a, \varepsilon) = \{(q_1, \underline{A})\},$$

$$\delta(q_1, a, \varepsilon) = \{(q_1, A)\},$$

$$\delta(q_1, b, A) = \{(q_2, \varepsilon)\},$$

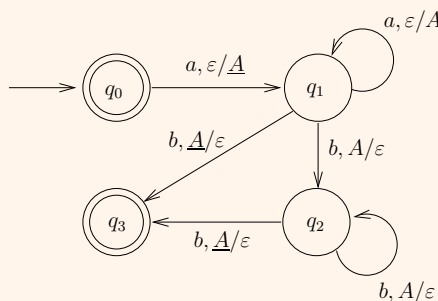
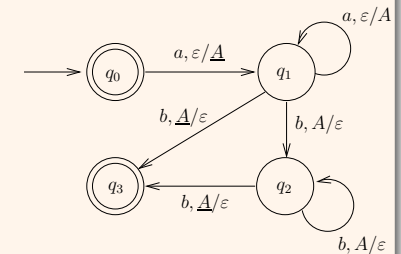
$$\delta(q_1, b, \underline{A}) = \{(q_3, \varepsilon)\},$$

$$\delta(q_2, b, A) = \{(q_2, \varepsilon)\},$$

$$\delta(q_2, b, \underline{A}) = \{(q_3, \varepsilon)\},$$

$$\delta(q, \sigma, \gamma) = \emptyset \quad \text{for other } (q, \sigma, \gamma).$$

Diagram representation:



A computation of the automaton on input $aabb$:

$$\begin{array}{lcl} (q_0, aabb, \varepsilon) & \vdash & (q_1, abb, \underline{A}) \vdash (q_1, bb, \underline{AA}) \\ & \vdash & (q_2, b, \underline{A}) \vdash (q_3, \varepsilon, \varepsilon). \end{array}$$

Since $q_3 \in F = \{q_0, q_3\}$, we have that $aabb \in \mathcal{L}(M)$.

7.5 Pushdown automata and context-free languages

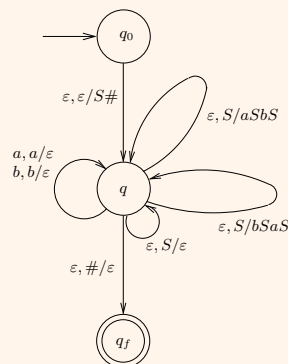
Theorem 7.4

A language is context-free if and only if it can be recognised by a (nondeterministic) pushdown automaton.

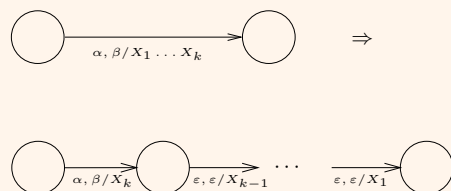
- The proof of the direction “if L can be recognised by a (nondeterministic) pushdown automaton, then L is context-free” is omitted here but can be found in Sipser’s book (Lemma 2.27).
- Also the detailed proof of the direction “if L is context-free, then L can be recognised by a (nondeterministic) pushdown automaton” is omitted. The main idea, however, is to have the pushdown automaton M_G corresponding to a CFG G try out leftmost derivations $S \xRightarrow{\text{lm}}^* x$ of G on its stack as follows:
 - ▶ If the topmost symbol on the stack is a variable A , the machine chooses some rule $A \rightarrow \omega$ and pushes the symbols corresponding to ω on its stack.
 - ▶ If the topmost symbol on the stack is a terminal symbol a , the machine tries to match it with the current input symbol.

Example:

A pushdown automaton corresponding to the CFG $\{S \rightarrow aSbS \mid bSaS \mid \varepsilon\}$:



We use the following natural abbreviation in the transition diagram:



53/62

For instance, on input $abab$ the automaton has the accepting computation

$$\begin{array}{lll}
 (q_0, abab, \varepsilon) \vdash (q, abab, S\#) & \vdash^* & (q, abab, aSbS\#) \\
 & \vdash & (q, bab, SbS\#) \vdash^* (q, bab, bSaSbS\#) \\
 & \vdash & (q, ab, SaSbS\#) \vdash (q, ab, aSbS\#) \\
 & \vdash & (q, b, SbS\#) \vdash (q, b, bS\#) \\
 & \vdash & (q, \varepsilon, S\#) \vdash (q, \varepsilon, \#) \\
 & \vdash & (q_f, \varepsilon, \varepsilon).
 \end{array}$$

This corresponds to the following leftmost derivation of $abab$ in the grammar:

$$\begin{array}{lcl}
 \underline{S} & \Rightarrow & aSbS \\
 & \Rightarrow & abab\underline{S} \\
 & \Rightarrow & abab.
 \end{array}$$

- A pushdown automaton M is *deterministic*, if for each configuration (q, w, α) there is at most one configuration (q', w', α') such that

$$(q, w, \alpha) \vdash_M (q', w', \alpha').$$

- Unlike in the case of finite automata, *nondeterministic pushdown automata are properly more powerful than deterministic ones*.
- As an example, the language $\{ww^R \mid w \in \{a, b\}^*\}$ can be recognised by a nondeterministic pushdown automaton but not by any deterministic one. (Proof omitted.)
- A context-free language is *deterministic* if it can be recognised by some deterministic pushdown automaton.
- Deterministic context-free languages can be parsed in time $O(n)$; in general, parsing context-free languages with currently known algorithms takes almost time $O(n^3)$.

Limitations of Context-Free Languages

7.6 A pumping lemma for context-free languages

- There is a “pumping lemma” for context-free languages as well.
- However, now the string must be “pumped” synchronously in two parts.

Lemma 7.5 (The “uvwxy-lemma”)

Let L be a context-free language. Then for some $n \geq 1$, every string $z \in L$ with $|z| \geq n$ can be divided in five parts $z = uvwxy$ in such a way that

1. $|vx| \geq 1$,
2. $|vwx| \leq n$, and
3. $uv^iwx^iy \in L$ for all $i = 0, 1, 2, \dots$

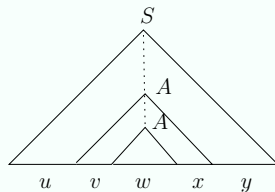
Proof

Let $G = (V, \Sigma, P, S)$ be a CFG, in Chomsky normal form, that generates the language L .

Then any parse tree of height h in G has at most 2^h leaf nodes. In other words, any parse tree for a string $z \in L$ contains a path (from the root to a leaf) whose length is at least $\log_2 |z|$.

Let k be the number of variables in G and define $n = 2^{k+1}$. Consider any string $z \in L$ with $|z| \geq n$, and take any parse tree for it.

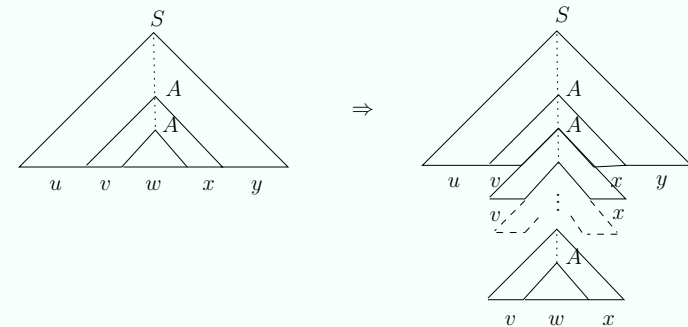
By the above observations, the tree has a path whose length is at least $k+1$; in such a path, at least one variable A must occur twice (in fact, already among the $k+2$ lowest nodes of the path).



The string z can now be divided in five parts $z = uvwxy$, where w is the substring derived from the lowest occurrence of A and vwx is the substring derived from the second-lowest occurrence of A in the path.

The substrings can thus be obtained from a derivation

$$S \Rightarrow^* uAy \Rightarrow^* uvAxy \Rightarrow^* uvwxy.$$



As $S \Rightarrow^* uAy$, $A \Rightarrow^* vAx$ and $A \Rightarrow^* w$, we can “pump” the substrings v and x around the substring w :

$$S \Rightarrow^* uAy \Rightarrow^* uvAxy \Rightarrow^* uv^2Ax^2y \Rightarrow^* \dots \Rightarrow^* uv^iAx^iy \Rightarrow^* uv^iwx^iy.$$

Thus $uv^iwx^iy \in L$ for all $i = 0, 1, 2, \dots$

As the CFG G is in Chomsky normal form and $A \Rightarrow^* vAx$, we necessarily have $|vx| \geq 1$.

And as we selected the variable A in a way that guarantees that its second-lowest occurrence is at most height $k + 1$ from the leaf nodes of the parse tree, the substring vwx derived in the parse tree rooted at this occurrence has at most $2^{k+1} = n$ symbols.

Example:

Consider the language

$$L = \{a^k b^k c^k \mid k \geq 0\}.$$

Suppose that L would be context-free and consider the string $z = a^n b^n c^n \in L$, where n is the length parameter in Lemma 7.5.

By the Lemma, we should now be able to divide z into five parts

$$z = uvwxy, \quad |vx| \geq 1, \quad |vwx| \leq n.$$

Due to the second condition above, the substring vx contains at least one symbol: a , b , or c . But due to the third condition, the substring vx cannot contain every symbol a , b and c . Therefore, the string $uv^0wx^0y = uwy$ cannot contain equal numbers of all symbols a , b and c , and thus does not belong to language L . Therefore, our assumption is wrong and L is not context-free.