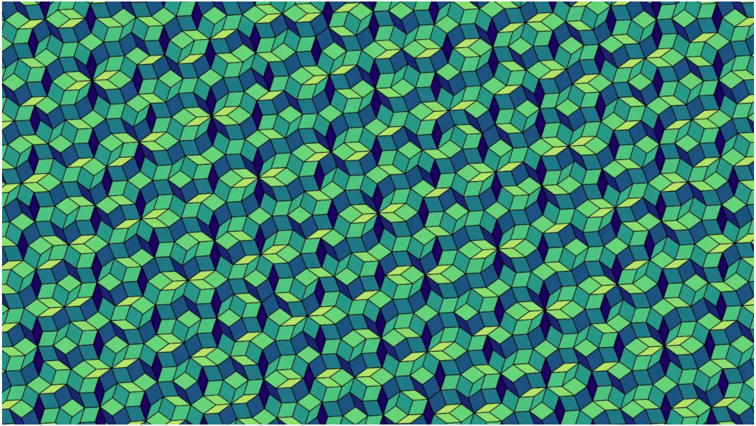
Automata, Languages and Computation

Chapter 7 : Properties of Context-Free Languages
Part II

Master Degree in Computer Engineering
University of Padua
Lecturer : Giorgio Satta

Lecture based on material originally developed by :
Gösta Grahne, Concordia University

Properties of Context-Free Languages



1. Pumping lemma for CFLs : similar to regular languages
2. Closure properties for CFL : some of the closure properties of regular languages also hold for CFLs
3. Computational properties for CFLs : we can efficiently implement previous transformations for CFGs and PDAs
4. Decision problems for CFLs : we can test emptiness and membership; equivalence and other problems are undecidable

Pumping lemma for CFLs

In each sufficiently long string of a CFL we can find two substrings "next to each other" that

- can be eliminated
- can be iterated (synchronously)

still resulting in strings of the language

This property can be used to prove that some languages are not CFL

Parse trees

Theorem Let G be some CFG in CNF. Let T be a parse tree for a string $w \in L(G)$. If the longest path in T has n arcs, then $|w| \leqslant 2^{n-1}$

Proof By induction on $n \geqslant 1$

Base $n = 1$. T has one leaf and one inner node (root), and represents a derivation $S \Rightarrow a$. We have $|w| = 1 \leqslant 2^{n-1} = 2^0 = 1$

Parse trees

Induction $n > 1$. T's root uses a production $S \to AB$, and we can write $S \Rightarrow AB \overset{*}{\Rightarrow} w = uv$, where $A \overset{*}{\Rightarrow} u$ and $B \overset{*}{\Rightarrow} v$

We are using factorization here

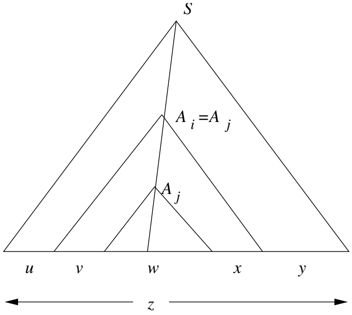
No path under the subtree rooted at A or B can have length greater than $n - 1$. By the inductive hypothesis we have $|u| \leqslant 2^{n-2}$ and $|v| \leqslant 2^{n-2}$

We can conclude that $|w| = |u| + |v| \leqslant 2^{n-2} + 2^{n-2} = 2^{n-1}$ $\square$

Pumping lemma for CFLs

Theorem Let L be some CFL. There exists a constant n such that, if $z \in L$ and $|z| \geqslant n$, we can factorize $z = uvwxy$ under the following conditions :

- $|vwx| \leqslant n$
- $vx \neq \epsilon$
- $uv^iwx^iy \in L$, for each $i \geqslant 0$



Pumping lemma for CFLs

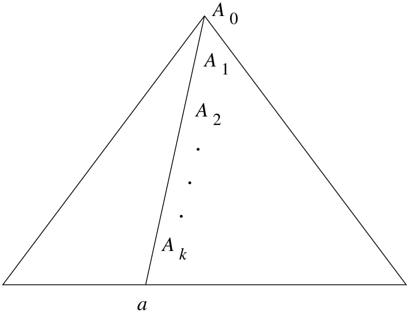
Proof Let G be some CFG in CNF such that $L(G) = L \setminus \{\epsilon\}$. Let m be the number of variables of G. We choose $n = 2^m$

Let $z \in L$ such that $|z| \geqslant n$

From a previous theorem, the parse tree for z must have some path of length greater than m, otherwise we would get $|z| \leqslant 2^{m-1} = n/2$

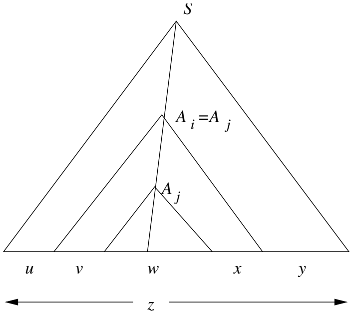
Pumping lemma for CFLs

Consider all occurrences of variables in a path of length $k+1$, where $k \geqslant m$



Pumping lemma for CFLs

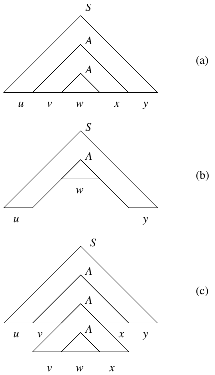
Since G has only m variables, at least one variable occurs more than once in the path. Let us assume $A_i = A_j$, where $k - m \leqslant i < j \leqslant k$, that is, we choose A_i in the lower part of the path



Pumping lemma for CFLs

We can then edit the parse tree in (a) in such a way that

- its yield becomes uv^0wx^0y, as shown in (b)
- its yield becomes uv^2wx^2y, as shown in (c)



Pumping lemma for CFLs

In the general case, we can edit the parse tree in (a) in such a way that its yield becomes uv^iwx^iy, for any $i \geqslant 0$

Since the longest path in the subtree rooted at A_i has length no longer than $m + 1$, a previous theorem allows us to assert that $|vwx| \leqslant 2^m = n$ $\square$

Example

Consider $L = \{0^i 1^i 2^i \mid i \geqslant 1\}$, and let n be the pumping lemma constant associated with L. We choose $z = 0^n 1^n 2^n$

For any factorization of z into $uvwxy$, with $|vwx| \leqslant n$ and v and x not both empty, we have that vwx cannot contain both 0 and 2, because the rightmost 0 and the leftmost 2 are $n + 1$ places away one from the other

We therefore have the following cases:

- vwx does not contain 2; then vx has only 0 and 1; then uwy, which should be in L, has n occurrences of 2 but less than n occurrences of 0 or 1
- vwx does not contain 0; a similar reasoning as in the first case applies

Consequences of the pumping lemma

A CFL cannot **count** in more than two sequences

Example : $L = \{0^i 1^i 2^i \mid i \geqslant 1\}$

See previous slide

Try also to recognize L with a PDA

Consequences of the pumping lemma

A CFL cannot generate **crossing pairs**

Example : $L = \{0^i 1^j 2^i 3^j \mid i, j \geqslant 1\}$

Given n, we choose $z = 0^n 1^n 2^n 3^n$. Then vwx covers occurrences of at most two alphabet symbols. In all possible factorizations, the strings generated by iteration do not belong to L

Consequences of the pumping lemma

A CFL cannot generate **string copies**

Example : $L = \{ww \mid w \in \{0,1\}^*\}$

Given n, we choose $z = 0^n 1^n 0^n 1^n$. In all possible factorizations, the strings generated by iteration do not belong to L

Exercise

Using the pumping lemma, prove that the language

$$L = \{a^i b^j c^k \mid i, j \geqslant 0,\ k = \max\{i, j\}\}$$

is not context-free

Exercise

Solution Let us assume that L is a CFL; we will establish a contradiction. Let n be the pumping lemma constant associated with L

We choose $z = a^n b^n c^n \in L$ and analyze all possible factorizations $z = uvwxy$ with $|vwx| \leqslant n$ and $vx \neq \epsilon$, looking for a factorization that satisfies the pumping lemma

Exercise

$$z = \underbrace{a \cdots\cdots a}_{a \text{ block}} \underbrace{b \cdots\cdots b}_{b \text{ block}} \underbrace{c \cdots\cdots c}_{c \text{ block}}$$

We distinguish the following cases

- vwx is placed into the a block or into the b block
- vwx is placed into the c block
- vwx is placed across the a and b blocks, or else across the b and c blocks
  - v or x contain both a and b, or both b and c
  - v is placed into the a block and x is placed into the b block
  - v is placed into the b block and x is placed into the c block

Exercise

vwx is placed into the a block : consider the new string uv^kwx^ky with $k > 1$, which must belong to L

$\#_a$ (the number of a's) increases ($> n$), since $vx \neq \epsilon$, while $\#_c$ remains unchanged ($= n$) and equal to $\#_b$, that is, the minimum between $\#_a$ and $\#_b$

We therefore conclude that $uv^kwx^ky \notin L$ for $k > 1$

A similar reasoning applies to the case where vwx is placed into the b block

Exercise

vwx is placed into the c block : consider the new string uv^kwx^ky with $k = 0$, which must belong to L

$\#_c$ decreases ($< n$), since $vx \neq \epsilon$, and is no longer equal to the maximum between $\#_a$ and $\#_b$, which is n, since the a block and the b block both remain unchanged

We therefore conclude that $uv^kwx^ky \notin L$ for $k = 0$

Exercise

vwx is placed across the a and b blocks or else across the b and c blocks

- v or x include both a and b : choosing $k = 2$, we break the structure $a^* b^* c^*$ and the new string doesn't belong to L
- v or x include both b and c : we use the same argument of the previous point
- v is placed into the a block and x is placed into the b block : choosing $k = 2$, increases $\#_a$ and/or $\#_b$ $(> n)$, while $\#_c$ remains unchanged $(= n)$ and therefore will not be equal to the maximum required; therefore the new string does not belong to L

Exercise

vwx is placed across the a and b blocks or else across the b and c blocks (continued)

- v is placed into the b block and x is placed into the c block
  - if $x \neq \epsilon$ we choose $k = 0$; $\#_c$ becomes smaller (and so does $\#_b$ if $v \neq \epsilon$) but $\#_a$ does not change, and provides the maximum value; therefore $uv^k wx^k y \notin L$ for $k = 0$
  - if $x = \epsilon$ we choose $k > 1$ so that $\#_b$ gets larger than $\#_a$, and $\#_c$ does not change; therefore $uv^k wx^k y \notin L$ for some appropriate $k > 1$

Exercise

In none of the possible cases we have been able to satisfy the pumping lemma: we have established a **contradiction**

We then conclude that language L is not CFL

Substitution

Assume two (finite) alphabets Σ and Δ, and a function

$$s : \Sigma \to 2^{\Delta^*}$$

Let $w \in \Sigma^*$, with $w = a_1 a_2 \cdots a_n$, $a_i \in \Sigma$. We define

$$s(w) = s(a_1).s(a_2). \cdots .s(a_n)$$

and, for $L \subseteq \Sigma^*$, we define

$$s(L) = \bigcup_{w \in L} s(w)$$

Function s is called a **substitution**

Example

Let $s(0) = \{a^n b^n \mid n \geqslant 1\}$ and $s(1) = \{aa, bb\}$

Then $s(01)$ is a language whose strings have the form $a^n b^n aa$ or $a^n b^{n+2}$, with $n \geqslant 1$

Let $L = L(\mathbf{0}^*)$. Then $s(L)$ is a language whose strings have the form

$$a^{n_1} b^{n_1} a^{n_2} b^{n_2} \cdots a^{n_k} b^{n_k},$$

with $k \geqslant 0$ and with $n_1, n_2, \ldots, n_k$ positive integers

Substitution

Next theorem is used later to prove several closure properties for CFL in a unified way and through very simple proofs

Theorem Let L be a CFL defined over Σ and let s be a substitution defined on Σ such that, for each $a \in \Sigma$, $s(a)$ is a CFL. Then $s(L)$ is a CFL

Proof Let $G = (V, \Sigma, P, S)$ be a CFG generating L and, for each $a \in \Sigma$, let $G_a = (V_a, T_a, P_a, S_a)$ be a CFG generating $s(a)$

Substitution

We construct a CFG $G' = (V', T', P', S)$ with

$$V' = (\bigcup_{a \in \Sigma} V_a) \cup V$$

$$T' = \bigcup_{a \in \Sigma} T_a$$

$$P' = (\bigcup_{a \in \Sigma} P_a) \cup P_R$$

where P_R is obtained from P by replacing each occurrence of a in any right-hand side with symbol S_a

Substitution

We prove $L(G') = s(L)$

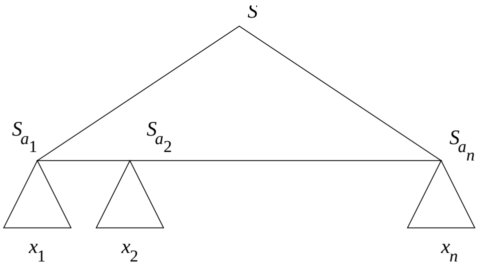
(Part $\supseteq$) Let $w \in s(L)$. Then there exists a string $x \in L$ such that

$$x = a_1 a_2 \cdots a_n$$

Furthermore, there exist strings $x_i \in s(a_i)$, such that
$w = x_1 x_2 \cdots x_n$

Substitution

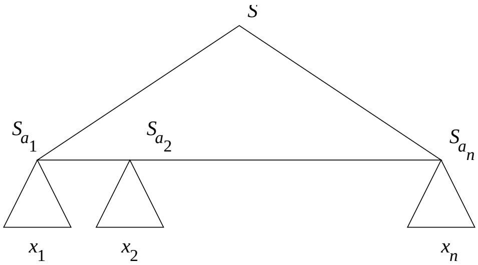
The associated parse tree for G' must have the form



We can then generate $S_{a_1} S_{a_2} \cdots S_{a_n}$ in G', and then generate $x_1 x_2 \cdots x_n = w$. Therefore $w \in L(G')$

Substitution

(Part $\subseteq$) Let $w \in L(G')$. Then the parse tree for w must have the form



Substitution

We can remove the subtrees at the bottom, and get a parse tree with yield

$$S_{a_1} S_{a_2} \cdots S_{a_n}$$

corresponding to a string $a_1 a_2 \cdots a_n \in L(G)$

We must also have $w \in s(a_1 a_2 \cdots a_n)$, and thus $w \in s(L)$ $\square$

Applications of the substitution theorem

Theorem The CFLs are closed under the following operations

- union
- concatenation
- Kleene closure ($*$) and positive closure ($+$)
- homomorphism

Proof For each of the operators above, we define a specific substitution and we apply the previous theorem

Union : Given two CFLs L_1 and L_2, consider the CFL $L = \{1, 2\}$. and define $s(1) = L_1$, $s(2) = L_2$. We have $L_1 \cup L_2 = s(L)$, which still is a CFL

Applications of the substitution theorem

Concatenation : Given two CFLs L_1 and L_2, consider the CFL $L = \{1.2\}$ and define $s(1) = L_1$, $s(2) = L_2$. We thus have $L_1.L_2 = s(L)$, which still is a CFL

$*$ *and* $+$ *closures* : Given a CFL L_1, consider the CFL $L = \{1\}^*$ and define $s(1) = L_1$. We have $L_1^* = s(L)$, which still is a CFL. A similar argument holds for $+$

Homomorphism : Assume a CFL L and a homomorphism h, both over Σ. We define $s(a) = \{h(a)\}$ for each $a \in \Sigma$. We then have $h(L) = s(L)$, which still is a CFL $\square$

Closure under string reverse

Theorem If L is a CFL, then so is L^R

Proof Assume L is generated by a CFG $G = (V, T, P, S)$. We build $G^R = (V, T, P^R, S)$, where

$$P^R = \{A \to \alpha^R \mid (A \to \alpha) \in P\}$$

Using induction on derivation length in G and in G^R, we can show that $(L(G))^R = L(G^R)$ (omitted) $\square$

CFL & intersection

$L_1 = \{0^n 1^n 2^i \mid n \geqslant 1,\ i \geqslant 1\}$ is a CFL, generated by the CFG

$$\begin{aligned} S &\to AB \\ A &\to 0A1 \mid 01 \\ B &\to 2B \mid 2 \end{aligned}$$

$L_2 = \{0^i 1^n 2^n \mid n \geqslant 1,\ i \geqslant 1\}$ is a CFL, generated by the CFG

$$\begin{aligned} S &\to AB \\ A &\to 0A \mid 0 \\ B &\to 1B2 \mid 12 \end{aligned}$$

$L_1 \cap L_2 = \{0^n 1^n 2^n \mid n \geqslant 1\}$ which is not a CFL

This was proved in a previous example

Intersection between CFL and regular language

Theorem Let L be some CFL and let R be some regular language. Then $L \cap R$ is a CFL

Proof Let L be accepted by the PDA

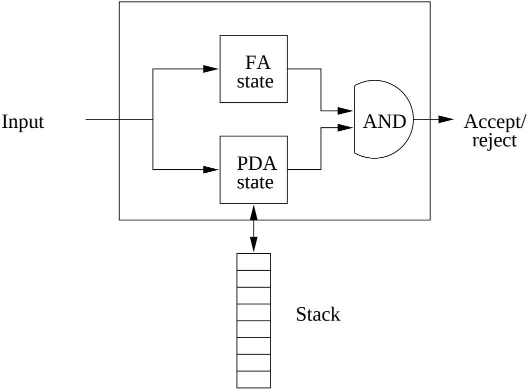
$$P = (Q_P, \Sigma, \Gamma, \delta_P, q_P, Z_0, F_P)$$

by final state, and let R be accepted by the DFA

$$A = (Q_A, \Sigma, \delta_A, q_A, F_A)$$

Intersection between CFL and regular language

We construct a PDA for $L \cap R$ based on the following idea



Intersection between CFL and regular language

We define

$$P' = (Q_P \times Q_A, \Sigma, \Gamma, \delta, (q_P, q_A), Z_0, F_P \times F_A)$$

where ($a \in \Sigma \cup \{\epsilon\}$)

$$\delta((q, p), a, X) = \{((r, s), \gamma) \mid (r, \gamma) \in \delta_P(q, a, X), s = \hat{\delta}_A(p, a)\}$$

We can show (omitted) by induction on the number of steps in the computation $\overset{*}{\vdash}$ that

$$(q_P, w, Z_0) \underset{P}{\overset{*}{\vdash}} (q, \epsilon, \gamma)$$

if and only if

$$((q_P, q_A), w, Z_0) \underset{P'}{\overset{*}{\vdash}} ((q, p), \epsilon, \gamma)\,, \text{ with } p = \hat{\delta}(q_A, w)$$

Intersection between CFL and regular language

(q, p) is an accepting state of P' if and only if

- q is an accepting state of P
- p is an accepting state of A

Therefore P' accepts w if and only if both P and A accept w, that is, $w \in L \cap R$ $\square$

Other properties for CFLs

Theorem Let L, L_1, L_2 be CFLs and let R be a regular language. Then

- $L \setminus R$ is a CFL
- $\overline{L}$ may fall outside of CFLs
- $L_1 \setminus L_2$ may fall outside of CFLs

Proof

Operator $\setminus$ *with REG* : $\overline{R}$ is regular, $L \cap \overline{R}$ is CFL, and $L \cap \overline{R} = L \setminus R$

Other properties for CFLs

Complement operator : If $\overline{L}$ would always be a CFL, then we have that

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

would always be CFL, which is a contradiction

Operator $\smallsetminus$ with CFL : Σ^* is a CFL. If $L_1 \smallsetminus L_2$ would always be a CFL, then $\Sigma^* \smallsetminus L = \overline{L}$ would always be a CFL, which is a contradiction $\square$

Test

Assert whether the following statements hold, and motivate your answer

- the intersection of a non-CFL L_1 and a CFL L_2 can be a non-CFL
- the intersection of a non-CFL and a finite language is always a CFL

Computational properties for CFLs

We investigate the **computational complexity** for some of the transformations previously presented

We need these results to establish the efficiency of some decision problems which we will consider later

We denote with n the **length** of the entire representation of a PDA or a CFG (for more detailed results, we should instead distinguish between number of variables, number of stack symbols, etc.)

Computational properties for CFLs

The following conversions can be computed in time $\mathcal{O}(n)$

- conversion from PDA accepting by final state to PDA accepting by empty stack
- conversion from PDA accepting by empty stack to PDA accepting by final state
- conversion from CFG to PDA

Given a PDA of size n we can build an equivalent CFG in time (and space) $\mathcal{O}(n^3)$, using a **preliminary binarization** of the transitions of the autmaton

The construction of Chapter 6 (which we have not presented) requires exponential time

Conversion to CNF

We can compute in time $\mathcal{O}(n)$

- the set of reachable symbols $r(G)$
- the set of generating symbols $g(G)$
- the elimination of useless symbols from a CFG

Conversion to CNF

We can compute in time $\mathcal{O}(n)$ the set of nullable symbols $n(G)$

We can compute in time $\mathcal{O}(n)$ the elimination of ϵ-productions from a CFG, using a **preliminary binarization** of the grammar

We can compute in time $\mathcal{O}(n^2)$ the set of unary symbols $u(G)$ and the elimination of unary productions from a CFG

Conversion to CNF

We can compute in time $\mathcal{O}(n)$ the replacement of terminal symbols with variables (first transformation for CNF)

We can compute in time $\mathcal{O}(n)$ the reduction of production with right-hand side length larger than 2 (second transformation for CNF)

Given a CFG of size n, we can construct an equivalent CFG in CNF in time (and space) $\mathcal{O}(n^2)$

Emptiness test

Let G be some CFG with start symbol S. $L(G)$ is empty if and only if S is not generating

We can then test emptiness for $L(G)$ using the already mentioned algorithm for the computation of $g(G)$, running in time $\mathcal{O}(n)$

CFL membership

The **membership problem** for a CFL string is defined as follows

Given as input a string w, we want to decide whether $w \in L(G)$, where G is some **fixed** CFG

Note : G does not depend on w and is **not** considered part of the input for our problem. Therefore the length of G does not affect the running time of the problem

CFL membership

Assume G in CNF and $|w| = n$. Since the parse trees for w are binary, the number of internal nodes for each tree is $2n - 1$ (proof by induction)

We can therefore generate all the parse trees of G with $2n - 1$ nodes and test whether some tree yields w

There are more efficient algorithms that take advantage of **dynamic programming** techniques

CFL membership

Let $w = a_1 a_2 \cdots a_n$. We construct a triangular **parse table** where cell X_{ij} is set valued and contains all variables A such that

$$A \overset{*}{\underset{G}{\Rightarrow}} a_i a_{i+1} \cdots a_j$$

X_{15}				
X_{14}	X_{25}			
X_{13}	X_{24}	X_{35}		
X_{12}	X_{23}	X_{34}	X_{45}	
X_{11}	X_{22}	X_{33}	X_{44}	X_{55}
a_1	a_2	a_3	a_4	a_5

CFL membership

We **iterately** construct the parse table, one row at a time and from bottom to top

First row is populated with the base case, while remaining rows are populated by the inductive case

Idea : $A \overset{*}{\underset{G}{\Rightarrow}} a_i a_{i+1} \cdots a_j$ if and only if

- for some production $A \to BC$
- for some integer k with $i \leqslant k < j$

we have $B \overset{*}{\underset{G}{\Rightarrow}} a_i a_{i+1} \cdots a_k$ and $C \overset{*}{\underset{G}{\Rightarrow}} a_{k+1} a_{k+2} \cdots a_j$

CFL membership

Base $X_{ii} \leftarrow \{A \mid (A \to a_i) \in P\}$

Induction We build X_{ij} for increasing values of $j - i \geqslant 1$

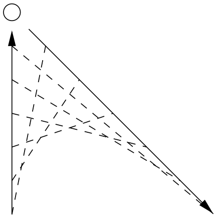
$X_{ij} \leftarrow X_{ij} \cup \{A\}$ if and only if there exist k, B, C such that

- $i \leqslant k < j$
- $(A \to BC) \in P$
- $B \in X_{ik}$ and $C \in X_{k+1,j}$

CFL membership

In the inductive case, to populate X_{ij} we need to check at most n pairs of previously built cells of the parse table

$$(X_{ii}, X_{i+1,j}),\ (X_{i,i+1}, X_{i+2,j}),\ \ldots,\ (X_{i,j-1}, X_{jj})$$



The operation above is related to vector convolution

CFL membership

We assume we can compute each check $B \in X_{ik}$ in time $\mathcal{O}(1)$. Then each set X_{ij} can be populated in time $\mathcal{O}(n)$

We need to populate $\mathcal{O}(n^2)$ sets X_{ij}

We summarize all of the previous observations by means of the following statement

Theorem The algorithm for the construction of the parse table computes all of the sets X_{ij} in time $\mathcal{O}(n^3)$. We then have $w \in L(G)$ if and only if $S \in X_{1n}$

Example

Let G be a CFG with productions

$$S \rightarrow AB \mid BC$$
$$A \rightarrow BA \mid a$$
$$B \rightarrow CC \mid b$$
$$C \rightarrow AB \mid a$$

and let $w = baaba$

$\{S,A,C\}$				
-	$\{S,A,C\}$			
-	$\{B\}$	$\{B\}$		
$\{S,A\}$	$\{B\}$	$\{S,C\}$	$\{S,A\}$	
$\{B\}$	$\{A,C\}$	$\{A,C\}$	$\{B\}$	$\{A,C\}$
b	a	a	b	a

Summary of decision problem for CFLs

We have presented **efficient** algorithms for the solution of the following decision problems for CFLs

- given a CFG G, test whether $L(G) \neq \emptyset$
- given a string w, test whether $w \in L(G)$ for a fixed CFG G

Undecidable decision problem for CFLs

In the next chapters we will develop a mathematical theory to prove the existence of decision problems that **no algorithm can solve**

Let us now anticipate some of these problems, concerning CFLs

- given a CFG G, test whether G is ambiguous
- given a representation for a CFL L, test whether L is inherently ambiguous
- given a representation for two CFLs L_1 and L_2, test whether the intersection $L_1 \cap L_2$ is empty
- given a representation for two CFLs L_1 and L_2, test whether $L_1 = L_2$
- given a representation for a CFL L defined over Σ, test whether $L = \Sigma^*$