Properties of context-free languages

We will study

1) Normal forms for CFGs (useful for proving properties of CFLs)
2) Expressive power ⇒ pumping lemma for CFLs
3) Closure and decision properties

Normal forms for CFGs

We look at how to simplify CFGs, while preserving the generated language.

- gain efficiency in parsing
- simplify proving properties

1) Eliminate useless symbols:

We say that $X \in V$ is useful if

$$S \Rightarrow^* \alpha X \beta \Rightarrow^* w \qquad \text{with } w \in V_T^* \quad \alpha, \beta \in V^*$$

Thus, a symbol is useless (not useful) if it does not participate in any derivation of strings of the language.
⇒ can be eliminated

Definition: $X \in V$ is generating if $X \Rightarrow^* w$, for $w \in V_T^*$

$X \in V$ is reachable if $S \Rightarrow^* \alpha X \beta$, for $\alpha, \beta \in V^*$

Hence, X is useful, if it is both generating and reachable

We identify useless symbols by

1) eliminating non-generating symbols and all their productions
2) —"— unreachable —"—

Note: right order is important

Example: $\begin{cases} S \to AB \mid b \\ A \to a \end{cases}$

· we eliminate unreachable symbols: all are reachable
· —"— non-generating —"— :

we eliminate B and $S \to AB$

$\Rightarrow$ we obtain: $S \to b$
$A \to a$

But, if we do it in right order:

1) Eliminate non-generating symbols: B and $S \to AB$
2) —"— unreachable —"— : A and $A \to a$

$\Rightarrow$ We obtain: $S \to b$

1) Eliminating non-generating symbols:

Recursively:

basis: mark all terminals as generating

induction: for each production $A \to X_1 \cdots X_k$
if all of $X_1, \dots, X_k$ are marked as generating
then mark A as generating

terminate: when no new generating symbol is found

Example: G_1: $\begin{cases} S \to AB \mid AC \mid CD \\ A \to BB \\ B \to AC \mid ab \\ C \to Ca \mid CC \\ D \to Bc \mid b \mid d \end{cases}$

$\{a, b, d\}$

$\{ - \cancel{A}, \cancel{C}, B, D\}$

$\{ \quad -\text{''}- \quad , A\}$

$\{ \quad -\text{''}- \quad , S\}$ $\Rightarrow$ C is non-generating

$\Rightarrow$ Remove C and all productions involving C

2) Eliminating unreachable symbols

Recursively

basis: mark S as reachable

induction: for each production $A \to X_1 \cdots X_k$

if A is marked as reachable

then mark $X_1, \ldots, X_k$ as reachable

terminate when no new reachable symbol is found

Example: G_2: $\begin{cases} S \to AB \\ A \to BB \\ B \to ab \\ D \to b \mid d \end{cases}$

$\{S\}$

$\{S, A, B\}$

$\{S, A, B, a, b\}$ $\Rightarrow$ D, d are unreachable

$\Rightarrow$ Remove D, d and all productions involving them

2) Eliminate ε-productions

ε-production: $A \to \varepsilon$ slows down parsing

Definition: $X \in V_N$ is nullable if $X \Rightarrow^* \varepsilon$

We first need to find all nullable symbols:

Recursively:

basis: if P contains $A \to \varepsilon$, then mark A as nullable

induction: for each production $A \to X_1 \cdots X_n$
if all of $X_1, \ldots, X_n$ are marked as nullable
then mark A as nullable

terminate when no new nullable symbol is found

Example: G_1:
$$\begin{cases} S \to ABC \mid BCB \\ A \to aB \mid a \\ B \to CC \mid b \\ C \to S \mid \varepsilon \end{cases}$$

$\{C\}$
$\{C, B\}$
$\{C, B, S\}$

Knowing the nullable symbols allows us to compensate for the elimination of ε-transitions

Example: in G_1, since B and C are nullable, we can derive

$S \Rightarrow^* BCB$, $S \Rightarrow^* CB$, $S \Rightarrow^* BC$, $S \Rightarrow^* BB$,
$S \Rightarrow^* C$, $S \Rightarrow^* B$, $S \Rightarrow^* \varepsilon$

Hence, if we eliminate $C \to \varepsilon$, we have to add direct productions for the above derivations.

Algorithm to eliminate ε-productions

1) Identify all nullable symbols
2) Replace each production $A \to X_1 \cdots X_k$
   by the set of all productions of the form $A \to \alpha_1 \cdots \alpha_k$
   where $\alpha_i = X_i$, if X_i is not nullable
   $\alpha_i = X_i$ or ε, if X_i is nullable

3) Remove all ε-productions

Example: for G_1

$$\begin{cases} S \to ABC \mid AB \mid AC \mid A \mid \\ \quad\quad BCB \mid BC \mid BB \mid CB \mid B \mid C \mid \varepsilon \\ A \to aB \mid a \\ B \to CC \mid C \mid \varepsilon \mid b \\ C \to S \mid \varepsilon \end{cases}$$

Finally, remove all ε-productions

Note: the grammar no longer generates ε. (this is unavoidable)

3) Eliminate unit productions

1/12/2004

Unit-production: $A \to B$ slows down parsing

Algorithm to eliminate unit-productions

1) Remove ε-productions
2) For all $A, B \in V_N$
   if $A \stackrel{*}{\Rightarrow} B$ and $B \to \alpha$ is not unit
   then add $A \to \alpha$
3) Eliminate all unit-productions

How do we find $A \Rightarrow^* B$?

Since we have no ε-productions: $A \Rightarrow^* B$ only if

$$A \Rightarrow B_1 \Rightarrow B_2 \Rightarrow \cdots \Rightarrow B_k \Rightarrow B$$

where all B_i s are distinct

Hence, $k \leq |V_N|$, and we can use reachability in directed graphs.

<u>Example</u>: $G_1 : \begin{cases} S \to A \mid B \\ A \to Sa \mid a \\ B \to S \mid b \end{cases}$

reachability: $S \Rightarrow^* A$, $S \Rightarrow^* B$

$B \Rightarrow^* S$, $B \Rightarrow^* A$

We get: $\begin{cases} S \to Sa \mid a \mid b \mid A \mid B \\ A \to Sa \mid a \\ B \to Sa \mid a \mid b \mid S \end{cases}$

Removing unit-productions $\begin{cases} S \to Sa \mid a \mid b \\ A \to Sa \mid a \\ B \to Sa \mid a \mid b \end{cases}$

Note: A, B are now unreachable, and hence useless

We have seen: removal of : useless symb., ε-prod, unit-prod. (7.7.)

Does the order of the steps matter?

Observation:

- removing useless : does not add productions at all
  (and therefore not ε-prod. or unit-prod)
- removing ε-prod: could add unit-prod
- removing unit-prod: - needs removing ε-prod first
  - could create useless symbols
  - cannot create ε-prod.

$\Rightarrow$ The right order for removal is

1) ε-productions
2) unit-productions
3) useless symbols : first non-generating
   then unreachable

Chomsky Normal Form

Definition: a CFG G is in Chomsky Normal (CNF) if all its productions are of the form

$$A \rightarrow a \qquad \text{with} \qquad a \in V_T$$
$$A \rightarrow BC \qquad\qquad A, B, C \in V_N$$

Theorem: given a CFG G with $\varepsilon \notin \mathcal{L}(G)$
there is a CNF grammar G' with $\mathcal{L}(G') = \mathcal{L}(G)$.

Proof : constructive , in 3 steps

1) Eliminate ε-prod. and unit-prod.

$\Rightarrow$ All productions are of the form

$$A \rightarrow a$$
$$A \rightarrow X_1 \cdots X_k \qquad (k \geq 2)$$

with $A \in V_N$, $a \in V_T$, $X_1, \dots, X_k \in V$

2) Remove "mixed bodies"

for each $a \in V_T$, add a new nonterminal V_a and production $V_a \rightarrow a$

in each production $A \rightarrow X_1 \cdots X_k$, replace a with V_a

$\Rightarrow$ All productions are of the form

$$A \rightarrow a$$
$$A \rightarrow A_1 \cdots A_k \qquad (k \geq 2)$$

with $a \in V_T$, $A, A_1, \dots, A_k \in V_N$

3) "Factor" long productions

for each $A \rightarrow A_1 \cdots A_k$ with $k \geq 3$

- add new nonterminals $B_1, \dots, B_{k-2}$
- replace $A \rightarrow A_1 \cdots A_k$

with
$$A \rightarrow A_1 B_1$$
$$B_1 \rightarrow A_2 B_2$$
$$\vdots$$
$$B_{k-2} \rightarrow A_{k-1} A_k$$

The grammar we get is in CNF by construction.
It is easy to show that the language is preserved

Example: G_1
$$\begin{cases} S \to ABB \mid ab \\ A \to Ba \mid ba \\ B \to aAbB \end{cases}$$

Step 1: nothing to do

Step 2:
$$\begin{cases} V_a \to a \\ V_b \to b \\ S \to ABB \mid V_a V_b \\ A \to B V_a \mid V_b V_a \\ B \to V_a A V_b B \end{cases}$$

Step 3:
$$V_a \to a$$
$$V_b \to b$$
$$\begin{cases} S \to A B_1 \mid V_a V_b \\ B_1 \to BB \end{cases}$$
$$A \to B V_a \mid V_b V_a$$
$$\begin{cases} B \to V_a C_1 \\ C_1 \to A C_2 \\ C_2 \to V_b B \end{cases}$$

Pumping lemma for CFLs:

Example: $L_1 = \{a^n b^n \mid n \geq 1\}$

$L_2 = \{a^n b^{n+m} c^m \mid n, m \geq 1\}$

$L_3 = \{a^n b^{2n} c^n \mid n \geq 1\}$

Which are CFLs?

- L_1 : $\{S \to ab \mid aSb$ $\Rightarrow$ L_1 is a CFL
- $L_2 = \{a^n b^n b^m c^m \mid n, m \geq 1\} =$

$= \{a^n b^n \mid n \geq 1\} \circ \{b^m c^m \mid m \geq 1\}$

$$\begin{cases} S \to AC \\ A \to ab \mid aAb \\ C \to bc \mid bCc \end{cases} \Rightarrow L_2 \text{ is a CFL}$$

- L_3 is not a CFL, because number of b's depends on its context, both to the left and to the right.

How can we prove that?

Pumping lemma for CFLs

Let L be a CFL

Then there exists a constant $m > 0$ such that

for all $z \in L$ with $|z| \geq m$

there exist a decomposition $z = uvwxy$ such that

$|vwx| \leq m$

$|vx| > 0$

for all $i \geq 0$, $uv^iwx^iy \in L$

Example: $L_3 = \{a^n b^{2n} c^n \mid n \geq 1\}$ is not a CFL

Assume L_3 is CFL

By P.L. there exists $m > 0$

Choose $z = a^m b^{2m} c^m$ $\quad (|z| \geq m)$

Take an arbitrary decomposition $z = uvwxy$
with $|vwx| \leq m$ and $|vx| > 0$

Choose $i = 0$, and claim $uv^0wx^0y = uwy \notin L$

We get a contradiction.

Let's show that $uwy \notin L$:

since $|vwx| \leq m$, either vwx has no a's or no c's

case 1: vwx has no a's

let $z' = uv^0wx^0y = uwy$

$\Rightarrow |z'| = |z| - \underbrace{|vx|}_{>0} = 4m - |vx| < 4m$

but z' has m a's, and since $|z'| < 4m$
we cannot have $z' = a^m \underbrace{b^{2m} c^m}_{<3m}$

$\Rightarrow z' \notin L$

case 2: vwx has no c's
similar

$\boxed{\text{Exercise 7.1}}$ Show that $L = \{a^n b^n c^n \mid n \geq 1\}$
is not a CFL.

$\boxed{\text{Exercise 7.2}}$ Show that $L = \{ww \mid w \in \{a, b\}^*\}$
is not a CFL. [Hint: Consider $a^n b^n a^n b^n$]

Proof of pumping lemma: (7.12)

Let L be a CFL and $G = (V_N, V_T, P, S)$ a CNF grammar for $L - \{\varepsilon\}$.

We exploit the following fact:

If in a parse tree T for CNF grammar G all paths from root to leaf are of length $\leq l$ then yield(T) has $\leq 2^{l-1}$ symbols

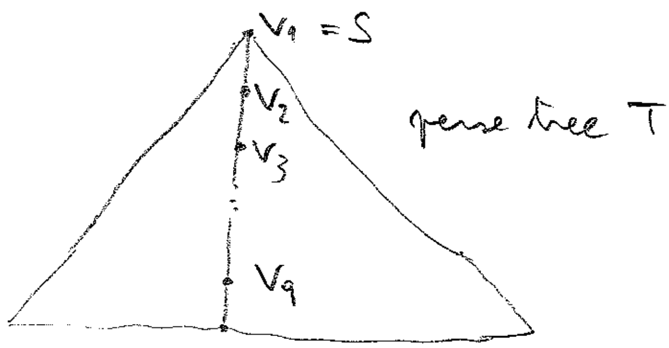
Intuitively: productions of CNF grammar have only two nonterminals on r.h.s.
$\Rightarrow$ T is a binary tree

Formally: by induction on l

Let $k = |V_N|$ and $n = 2^k$

Consider any $z \in \mathcal{L}(G)$ with $|z| > n = 2^k$

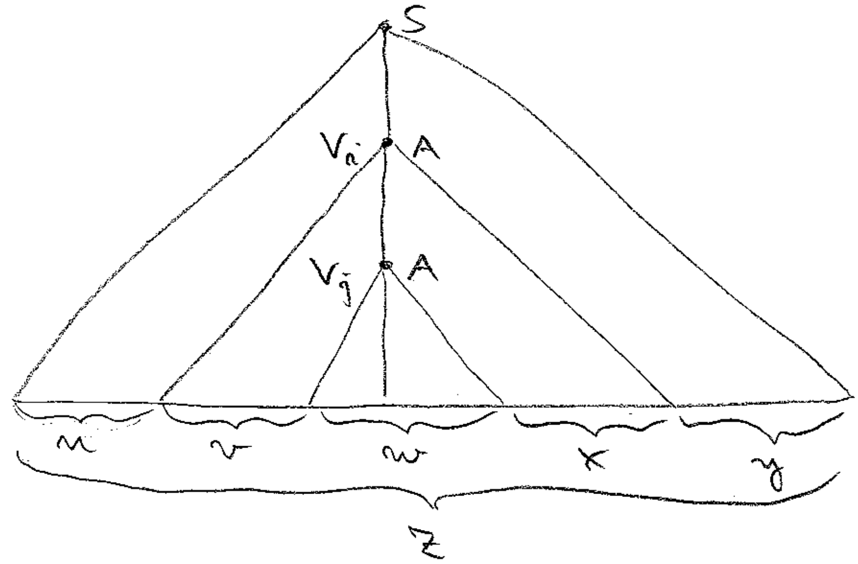
By fact, parse tree T for z has a root-leaf path Q of length $q \geq k+1$



Since $|V_N| = k < q$, by pigeon-hole principle, some non-terminal appears more than once on Q.

More precisely, $V_{q-k}, V_{q-k+1}, \ldots, V_{q-1}, V_q$ will contain a repetition

Let i, j be such that $q-k \leq i < j \leq q$ (7.13)
and $V_i = V_j = A$



We have $z = uvwxy$

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

Hence: $A \Rightarrow^* vAx$ and $A \Rightarrow^* w$

Therefore: $A \Rightarrow^* vAx \Rightarrow^* v^2Ax^2 \Rightarrow^* \cdots \Rightarrow^* v^iAx^i \Rightarrow^* v^iwx^i$

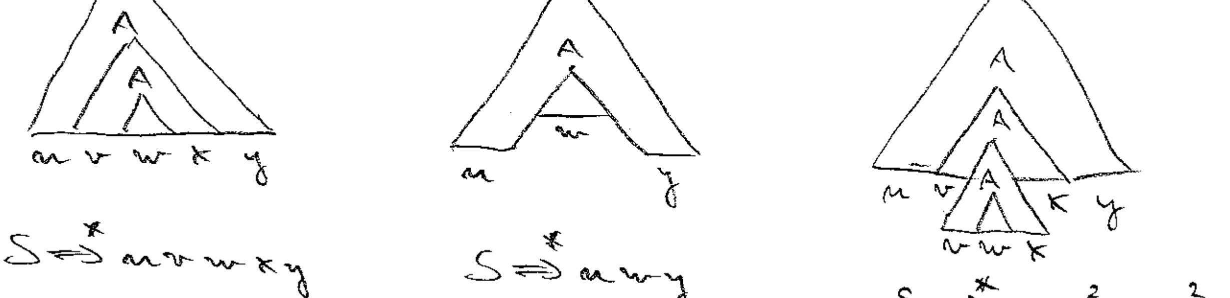
and also: $S \Rightarrow^* uAy \Rightarrow^* uv^iAx^iy \Rightarrow^* uv^iwx^iy$

or $S \Rightarrow^* uAy \Rightarrow^* uwy$

We get: $\forall i \geq 0 \quad S \Rightarrow^* uv^iwx^iy$

and hence $\forall i \geq 0 \quad uv^iwx^iy \in \mathcal{L}(G)$

In pictures



Moreover 1) $|vx| > 0$ since G has no ε-productions

2) $|vwx| \leq n = 2^k$ since $V_i = A$ is at height $\leq k+1$ and by fact has yield vwx of length $\leq 2^{(k+1)-1} = 2^k$. Q.e.d.

Closure properties of context-free languages

Fundamental operation: substitution $s(w)$ of a string w.

Consider an alphabet Σ; for every $a \in \Sigma$ we define a language L_a, on any alphabet (not necessarily Σ of course). We shall denote $L_a = s(a)$

Consider now a string $w = a_1 a_2 \cdots a_m \in \Sigma^*$; the substitution $s(w)$ of w is the language

$$\{ x_1 x_2 \cdots x_m \mid x_i \in s(a_i) \text{ for all } i \in \{1, \dots, m\} \}$$

which is the concatenation of all $s(a_i)$ for $i \in \{1, \dots, m\}$:

$$s(w) = s(a_1) \circ s(a_2) \circ \dots \circ s(a_m)$$

The definition extends to languages: given a language L, we define $s(L) = \bigcup_{w \in L} s(w)$.

Example

let $s(0) = \{a^n b^n \mid n \geq 1\}$
$s(1) = \{aa, bb\}$

s is a substitution on $\Sigma = \{0,1\}$.

Let $w = 01$; therefore $s(w) = s(0)s(1)$; we have

$$s(w) = \{a^n b^n aa \mid n \geq 1\} \cup \{a^n b^{n+2} \mid n \geq 1\}.$$

Theorem

Let L be a context-free language over an alphabet Σ, and s a substitution on Σ such that $s(a)$ is a context-free language for every $a \in \Sigma$. Then $s(L)$ is a context-free language.

Proof (sketch)

The idea is that we consider a context-free grammar for L, and we replace each terminal $a \in \Sigma$ with the start symbol (axiom) of the CFG for $s(a)$.
More formally, let $G = (V, \Sigma, P, S)$ the CFG for L and $G_a = (V_a, T_a, P_a, S_a)$ the CFG for each $a \in \Sigma$.
We choose disjoint variables (non-terminals) to avoid mixing the productions from different grammars.
The grammar $G' = (V', T', P', S)$ for $s(L)$ is as follows:

(i) V' is the union of V and all the V_a's for all $a \in \Sigma$

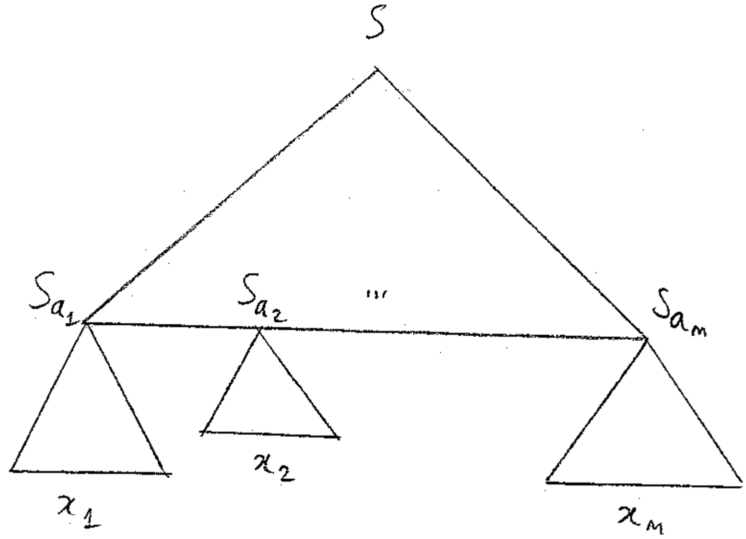
(ii) T' is the union of all T_a's for all $a \in \Sigma$

(iii) P' consists of

(1) all productions in all P_a's, for all $a \in \Sigma$

(2) the productions of G, where each terminal a has been replaced (in the body) by S_a.

The typical parse tree for G' starts like a parse-tree for G, but in the frontier, instead of having symbols of Σ, has start symbols of G_a's, from which parse-trees for the G_a's stem.



The formal proof that $\mathcal{L}(G') = s(L)$ is left to the reader (remember that we need to prove both inclusions $\mathcal{L}(G') \subseteq s(L)$ and $\mathcal{L}(G') \supseteq s(L)$).

We now show how the substitution theorem can be used for proving familiar closure properties we have already seen for regular languages.

Theorem The class of context-free languages is closed under:

(1) union
(2) concatenation
(3) closure ($*$) and positive closure ($+$).

Proof

(1) Let L_1 and L_2 be CFL's. Consider the language $L = \{1,2\}$ on $\Sigma = \{1,2\}$: if we define $s(1) = L_1$ and $s(2) = L_2$, then $L_1 \cup L_2 = s(L)$. Thesis follows from substitution theorem.

(2) Let L_1 and L_2 be CFL's. Consider the language $L = \{12\}$ on $\Sigma = \{1,2\}$ and $s(1) = L_1$, $s(2) = L_2$; we have $s(L) = L_1 \circ L_2$, and again thesis follows from substitution theorem.

7.17

(3) Similarly to the previous cases, if we choose $L = \{1\}^*$ and $S(1) = L_1$, we have $L_1^* = s(L)$; if $L = \{1\}^+$ and $s(1) = L_1$, then $L_1^+ = s(L)$. Again, the thesis follows from substitution theorem.

5/12/2005

Intersection with a regular language

Unlike regular languages, CFL's are not closed under intersection. The following example provides a proper counterexample.

We know that $L = \{0^n 1^n 2^n \mid n \geq 1\}$ is non a CFL.
On the contrary, the following languages are context-free:

$$L_1 = \{0^n 1^n 2^k \mid n \geq 1, i \geq 1\}$$
$$L_2 = \{0^k 1^n 2^n \mid n \geq 1, k \geq 1\}$$

A grammar for L_1 is in fact

$S \to AB$
$A \to 0A1 \mid 01$
$B \to 2B \mid 2$

and one for L_2 is

$S \to AB$
$A \to 0A \mid 0$
$B \to 1B2 \mid 12$

Also, PDA's accepting L_1 and L_2 can be constructed (left as exercise to the reader).

Now, notice that $L = L_1 \cap L_2$; in fact L_1 requires that the 0's are as many as the 1's, L_2 requires that the 1's are as many as the 2's, and L requires both properties.

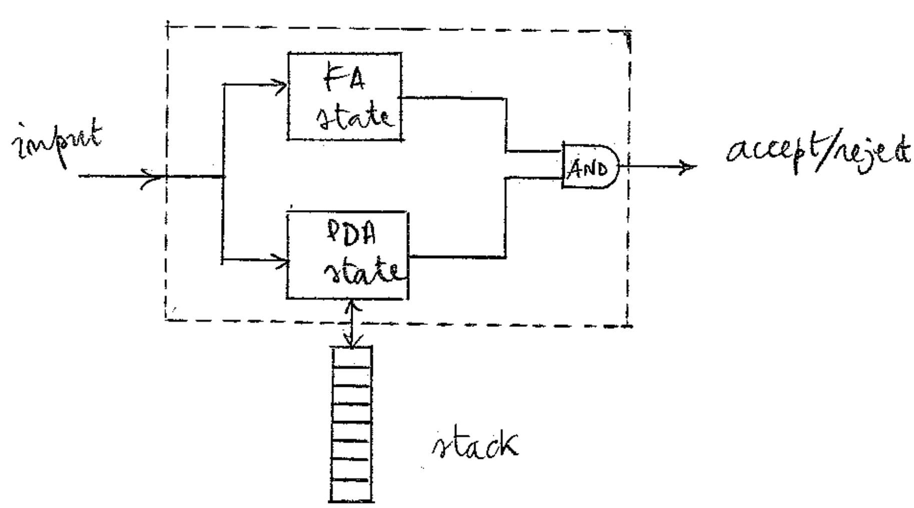
Now, if CFL's were closed under intersection, we could prove that L is CF, which is a contradiction. So CFL's are not closed under intersection.

On the contrary, if we intersect a CFL with a regular language, we obtain a CFL.

<u>Theorem</u> If L is a CFL and R is a regular language, then $L \cap R$ is a CFL.

NO

<u>Proof</u> The proof is obtained by making a PDA accepting L and a FA accepting R run in parallel; the overall automaton is a PDA, shown in figure:



7.19

The result is acceptance if both automata accept the input string.

Formally, let $P = (Q_P, \Sigma, \Gamma, \delta_P, q_P, Z_0, F_P)$ the PDA accepting L by final state, and

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

a DFA accepting R.

The PDA accepting $L \cap R$ is

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

with $\delta((q,p), a, X)$ is the set of all pairs $((r,s), \gamma)$ such that:

(i) $s = \delta_A(p, a)$ and

(ii) (r, γ) is in $\delta_P(q, a, X)$.

Intuitively, for each move of P, we make the same move in P, and at the same time we carry along the corresponding state of A.

The formal proof of the fact that $\mathcal{L}(P') = L \cap R$ is left to the reader.

↑ NO

<u>Theorem</u> Let L, L_1, L_2 be CFL's and R be a regular language; then we have

(1) $L - R$ is a CFL

(2) $\overline{L}$ is not necessarily a CFL

(3) $L_1 - L_2$ is not necessarily a CFL

Proof

(1) Observe that $L - R = L \cap \bar{R}$; $\bar{R}$ is regular because R is regular (closure property) ; from the theorem at page 7. we have the thesis.

(3) (2) By contradiction, assume that the complement of a CFLs is in general a CFL. Then

$$\overline{\bar{L}_1 \cup \bar{L}_2}$$

is a CFL. But we observe that $L_1 \cap L_2 = \overline{\bar{L}_1 \cup \bar{L}_2}$, which is not a CFL in general. Since this is a contradiction, we have the thesis.

(2) (3) the language Σ^* is known to be a CFL (it is easy to construct a CFG for it or a PDA accepting it). By contradiction, assume that the ~~difference~~ of two CFL's is a CFL. Then $\Sigma^* - L$ is in general a CFL. But $\Sigma^* - L = \bar{L}$, which in general is not a CFL ; this is a contradiction, so thesis follows.

Decision properties of context-free languages

Complexity of converting among CFG's and PDA's.

↓ NO

In the following we will consider, as size of a PDA or CFG, the entire length of the representation of the PDA or of the CFG.

The following conversions are <u>linear</u> in the input size (i.e. the time needed is a linear function of the input size):

(1) converting a CFG into a PDA

(2) converting a PDA accepting by final state to one accepting by empty stack

(3) converting a PDA accepting by empty stack to one accepting by final state.

Conversion to Chomsky normal form

Theorem Given a CFG G of length n, we can find an equivalent grammar in Chomsky normal form in time $O(n^2)$; the resulting grammar has length $O(n^2)$.

Proof Observe that:

(1) Detecting reachable and generating symbols can be done in time $O(n)$, so as elimination of useless symbols

(2) Construction of unit pairs takes $O(n^2)$; the resulting grammar has size $O(n^2)$

(3) The replacement of terminals by variables takes $O(n)$, resulting in a grammar of length $O(n)$

(4) Breaking productions with bodies longer than two takes $O(n)$ and results in a grammar of length $O(n)$.

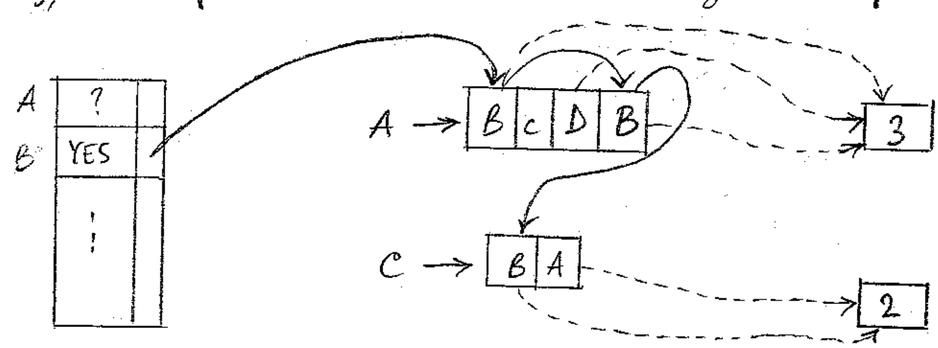
Unfortunately the elimination of ϵ-productions clearly requires time exponential in the length of the production bodies: however, we can apply this step only after having broken productions with body longer than 2. In this case the running time is $O(n)$ and the resulting grammar has length $O(n)$. This ends the proof.

Testing emptiness of a CFG

It is immediate to see that we can test emptiness of a CFG by checking whether the start symbol is generating.

A naive implementation of this test takes time $O(n^2)$, where n is the length of the grammar. In fact, each pass of the discovery of generating symbols can take $O(n)$, and there are $O(n)$ passes in the worst case.

However, a more careful algorithm can take time $O(n)$. Such algorithm makes use of a data structure which is built in advance. The data structure starts with an array indexed on the non-terminals (variables), and reports whether variables are generating or not:



For each variable, there is a chain of arrows linking positions in which such variable appears. In the figure, productions $A \to BcDB$ and $C \to BA$ are shown.
Moreover, dashed arrows suggest links from productions to the corresponding counts; each count represents the number of variables in the production for which the capability of producing terminals has not yet been verified.

If we discover that a variable is generating, we decrement every body where such variable appears by the number of positions in which it appears; for example, if B is discovered to be generating, we decrement the count of the production $A \to BcDB$ by 2.
When a count reaches 0, then the variable in the head is generating, and if it is not known to be generating, it is put on a queue (from the head of which variables are taken one by one).

We make the following observations:

(7.24)

(1) the creation and initialisation of the array takes time $O(n)$, since there are not more than n variables.

(2) Initialisations of links and counts takes $O(n)$, since there are at most n productions, with total length n.

(3) When a variable is discovered to be generating, there are two things to do:

(a) for each production: checking whether the count is 0, and putting the variable in the head in the queue if necessary; this takes $O(1)$ for each production, and therefore $O(n)$ overall;

(b) visiting the positions of the production bodies where the variable appears: work taking time proportional to the length of the production bodies, i.e. $O(n)$.

We can conclude that checking emptiness can be done in time $O(n)$.

7.25

Testing membership in a context-free language

Consider a string w, with $|w| = n$. A naïve algorithm for testing membership of w in a CFL L proceeds as follows. We construct a CFG G for L in Chomsky normal form. We notice that for a CFG derivation trees are binary, and a binary tree having n leaves (i.e., producing a word of length n) has $n-1$ non-terminal nodes. In fact, if we add a leave to a tree with n leaves, we need to add exactly ~~two~~ one non-leave node~~s~~. Therefore we can test membership of w in L by constructing all trees with $2n-1$ leaves; this of course takes time $O(2^n)$.

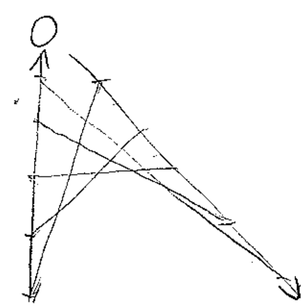
A much more clever algorithm is CYK (Cocke-Younger-Kasami), or table-filling algorithm.

We start from a grammar $G = (V, T, P, S)$ in Chomsky normal form. The input is

$$w = a_1 a_2 \cdots a_n \in T^*$$

We build a table as follows.

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	...



The table case X_{ij} is the set of variables $\{A \mid A \Rightarrow^* a_i a_{i+1} \cdots a_j\}$; so we are interested to know whether S appears in X_{1n}.

We work row by row from bottom to top.

BASE STEP (first row)

$X_{ii} = \{ A \mid A \to a_i$ is a production of $G \}$

INDUCTIVE STEP

To compute X_{ij} we exploit the X's in the row below, which tell us everything on all strings shorter than $a_i a_{i+1} \cdots a_j$.
Every derivation $A \Rightarrow^* a_i a_{i+1} \cdots a_j$ must start with a step of the form $A \Rightarrow BC$. Therefore, we have

$$B \Rightarrow^* a_i a_{i+1} \cdots a_k \quad \text{for some } k<j$$
$$C \Rightarrow^* a_{k+1} a_{k+2} \cdots a_j$$

So, in order to have A in X_{ij} we must find B and C in the lower rows and $k \geq 1$ such that:

(i) $i \leq k < j$
(ii) $B \in X_{ik}$
(iii) $C \in X_{k+1,j}$
(iv) $A \to BC$ is a production in G.

Complexity: the technique requires to compare at most n pairs of previously computed sets. There are $n(n+1)/2$ cells in the table, so the algorithm takes time $\Theta(n^3)$.

(Note: n is the length of the input string, while we consider the size of G to be constant!)

Example: $S \to AB \mid BC$
$A \to BA \mid a$
$B \to CC \mid b$
$C \to AB \mid a$

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

Undecidable CFL problems

As we will see in the following, there are some problems regarding CFL's that cannot be solved by an algorithm. This means that there may be an algorithm, but in some cases the execution of the algorithm may not terminate.

(1) is a given CFG ambiguous?

(2) is a given CFG inherently ambiguous?

(3) is the intersection of two CFL's empty?

(4) are two given CFL's equal?

(5) is a given CFL on an alphabet Σ equal to Σ^*?