

بسم الله الرحمن الرحيم

دانشگاه صنعتی اصفهان – دانشکده مهندسی برق و کامپیوتر
(نیم‌سال تحصیلی ۴۰۰۱)

نظریه زبان‌ها و ماشین‌ها

حسین فلسفین

Part 1: Removing Useless Productions

One invariably wants to remove productions from a grammar that can never take part in any derivation.

For example, in the grammar whose entire production set is

$$S \rightarrow aSb|\varepsilon|A, \quad A \rightarrow aA,$$

the production $S \rightarrow A$ clearly plays no role, as A cannot be transformed into a terminal string. While A can occur in a string derived from S , this can never lead to a sentence. Removing this production leaves the language unaffected and is a simplification by any definition.

Definition: Let $G = (V, \Sigma, R, S)$ be a context-free grammar. A variable $A \in V$ is said to be **useful** if and only if there is at least one $w \in L(G)$ such that

$$S \Rightarrow^* xAy \Rightarrow^* w,$$

with x, y in $(V \cup \Sigma)^*$. In words, a variable is useful if and only if it occurs in at least one derivation. A variable that is not useful is called **useless**. A production is useless if it involves any useless variable.

Evidently, omitting useless productions from a grammar will not change the language generated, so we may as well detect and eliminate all useless productions.

Two things a variable has to be able to do to be useful

1. We say a variable A is **generating** if $A \Rightarrow^* w$ for some terminal string w .
2. We say a variable A is **reachable** if there is a derivation $S \Rightarrow^* \alpha X \beta$ for some $\alpha \in (V \cup \Sigma)^*$ and $\beta \in (V \cup \Sigma)^*$.

Two reasons why a variable is useless: either because it cannot be reached from the start symbol or because it cannot derive a terminal string. A procedure for removing useless variables and productions is based on recognizing these two situations.

Surely a variable that is useful will be both generating and reachable. If we eliminate the variables that are **not generating first**, and then eliminate from the remaining grammar those variables that are **not reachable**, we shall, as will be proved, have only the useful symbols left.

Example 1: In a grammar with start symbol S and productions

$$S \rightarrow A, \quad A \rightarrow aA|\varepsilon, \quad B \rightarrow bA,$$

the variable B is useless and so is the production $B \rightarrow bA$. Although B can derive a terminal string, there is no way we can achieve $S \Rightarrow^ xBy$.*

Example 2: Eliminate useless symbols and productions from $G = (V, \Sigma, R, S)$, where $V = \{S, A, B, C\}$ and $\Sigma = \{a, b\}$, with R consisting of

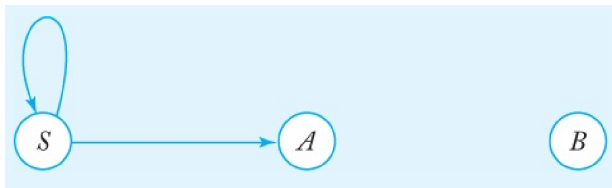
$$S \rightarrow aS|A|C, A \rightarrow a, B \rightarrow aa, C \rightarrow aCb.$$

First, we identify the set of variables that can lead to a terminal string. Because $A \rightarrow a$ and $B \rightarrow aa$, the variables A and B belong to this set. So does S , because $S \Rightarrow A \Rightarrow a$. However, this argument cannot be made for C , thus identifying it as useless. Removing C and its corresponding productions, we are led to the grammar G_1 with variables $V_1 = \{S, A, B\}$, terminals $\Sigma_1 = \{a\}$, and productions

$$S \rightarrow aS|A, A \rightarrow a, B \rightarrow aa.$$

ادامه مثال ۲

Next, we want to eliminate the variables that cannot be reached from the start variable. For this, we can draw a **dependency graph** for the variables. Dependency graphs are a way of visualizing complex relationships and are found in many applications. For CFGs, a dependency graph has its vertices labeled with variables, with an edge between vertices C and D if and only if there is a production of the form $C \rightarrow xDy$. A dependency graph for V_1 is shown in the following figure.



ادامه مثال ۲

A variable is useful only if there is a path from the vertex labeled S to the vertex labeled with that variable. In our case, the figure shows that B is useless. Removing it and the affected productions and terminals, we are led to the final answer with productions

$$S \rightarrow aS|A, A \rightarrow a.$$

بیان روند حذف متغیرها و رول‌های بی‌فایده

Let $G = (V, \Sigma, R, S)$ be a context-free grammar. Then there exists an equivalent grammar $\hat{G} = (\hat{V}, \hat{\Sigma}, \hat{R}, S)$ that does not contain any useless variables or productions.

The grammar \hat{G} can be generated from G by an algorithm consisting of two parts.

In the first part, we construct an intermediate grammar $G_1 = (V_1, \Sigma_1, R_1, S)$ such that V_1 contains only variables A for which $A \Rightarrow^* w \in \Sigma^*$ is possible. The steps in the algorithm are

Construction of the set of variables that derive terminal strings:

1. Set V_1 to \emptyset .
2. Repeat the following step until no more variables are added to V_1 . For every $A \in V$ for which R has a production of the form $A \rightarrow \alpha$ and $\alpha \in (V_1 \cup \Sigma)^*$, add A to V_1 . (Note that this rule includes the case where $\alpha = \varepsilon$; all variables that have ε as a production body are surely generating.)
3. Take R_1 as all the productions in R whose symbols are all in $(V_1 \cup \Sigma)$.

*In the second part of the construction, we get the final answer \hat{G} from G_1 . We draw the **variable dependency graph** for G_1 and from it find all variables that cannot be reached from S . These are removed from the variable set, as are the productions involving them. We can also eliminate any terminal that does not occur in some useful production.*

برای گام دوم، راه دیگر، بهره‌گیری از روند استقرایی زیر است (مثل الگوریتم قبل):
An inductive algorithm by which we can find the set of reachable symbols for the grammar $G = (V_1, \Sigma_1, R_1, S)$:

BASIS: S is surely reachable.

INDUCTION: Suppose we have discovered that some variable $A \in V_1$ is reachable. Then for all productions with A in the head, all the symbols of the bodies of those productions are also reachable.

Example 3:

$$S \rightarrow AC|BS|B$$

$$A \rightarrow aA|aF$$

$$B \rightarrow CF|b$$

$$C \rightarrow cC|D$$

$$D \rightarrow aD|BD|C$$

$$E \rightarrow aA|BSA$$

$$F \rightarrow bB|b$$

در گام اول، همه متغیرهای غیرسازنده باید حذف شوند:

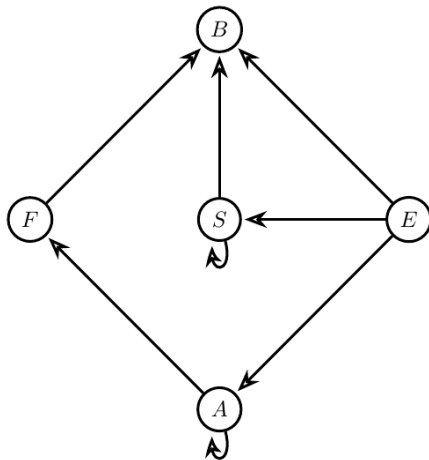
$$\{B, F\} \Rightarrow \{B, F, A, S\} \Rightarrow \{B, F, A, S, E\} \Rightarrow \{B, F, A, S, E\}$$

$$V_1 = \{S, A, B, E, F\}$$

$$\Sigma_1 = \{a, b\}$$

$$R_1 = \{S \rightarrow BS|B, A \rightarrow aA|aF, B \rightarrow b, E \rightarrow aA|BSA, F \rightarrow bB|b\}$$

در گام دوم، همه متغیرهای غیرقابل دسترسی باید حذف شوند:



گرامر نهایی \hat{G} ، فقط دارای دو رول $S \rightarrow SB|B$ و $B \rightarrow b$ است. نیز داریم $L(\hat{G}) = L(b^+)$.

نکته نهایی: حتماً ابتدا باید غیرسازنده‌ها را حذف کرد، سپس غیرقابل دسترس‌ها را. اگر برعکس عمل شود، ممکن است اشکال پیش آید؛ مانند مثال زیر.

Example 4: Consider the grammar $S \rightarrow AB|a$, $A \rightarrow b$. All variables but B are generating; S generates a , and A generates b . If we eliminate B , we must eliminate the production $S \rightarrow AB$, leaving the grammar $S \rightarrow a$, $A \rightarrow b$. Now we find that only S is reachable from S . Eliminating A and b leaves only the production $S \rightarrow a$. That production by itself is a grammar whose language is $\{a\}$, just as is the language of the original grammar.

Note that if we start by checking for reachability first, we find that all symbols of the grammar

$$S \rightarrow AB|a, A \rightarrow b$$

are reachable. If we then eliminate the symbol B because it is not generating, we are left with a grammar that still has useless symbols: $S \rightarrow a$, $A \rightarrow b$.


Part 2: Preview of Undecidable CFL Problems


In the next sessions we shall develop a remarkable theory that lets us prove formally that **there are problems we cannot solve by any algorithm that can run on a computer.** We shall use it to show that **a number of simple-to-state questions about grammars and CFL's have no algorithm; they are called "undecidable problems."** For example, the **following are undecidable:**


1. Is a given CFG G ambiguous? ($AMBIG_{CFG}$)
2. Is a given CFL inherently ambiguous?
3. Is the intersection of two CFL's empty?
4. Are two CFL's the same? (EQ_{CFG})
5. Is a given CFL equal to Σ^* , where Σ is the alphabet of this language? (ALL_{CFG})



Part 3: Some **Decidable** Properties of Context-Free Languages

 **A_{CFG} :** Given a string w of terminals, we want to know whether or not w is in $L(G)$. An algorithm that can tell us whether w is in $L(G)$ is a **membership algorithm**. We will establish the existence of a membership algorithm for context-free languages.

 **E_{CFG} :** Given a context-free grammar G , there exists an algorithm deciding **whether or not $L(G)$ is empty**.

 **$INFINITE_{CFG}$:** Given a context-free grammar G , there exists an algorithm for determining **whether or not $L(G)$ is infinite**.

Part 4: The CYK (Cocke-Younger-Kasami) Algorithm

In general, for a given language L , the problem of deciding whether a string w belongs to the language or not is called the **membership problem**.

Throughout this part, a context-free language is assumed to be given in terms of a context-free grammar $G = (V, \Sigma, R, S)$ in Chomsky normal form.

چرا رویکرد **brute-force** برای حل این مسئله بسیار ناکارآمد است؟

In order to generate a string of length n , we apply rules of the form $A \rightarrow BC$, $n - 1$ times and rules of the form $A \rightarrow a$, n times. So we apply rules $2n - 1$ times in total to generate a string of length n . Hence, given a string w of length n , we enumerate all the sequences of $2n - 1$ rules $r_1 r_2 \cdots r_{2n-1}$ and check to see if $S \Rightarrow^ w$ for each of the sequences. If $S \Rightarrow^* w$ for at least one of these sequences, the grammar generates w , and does not generate w otherwise. This is a typical example of a brute-force search. When the number of rules in the grammar is denoted by m , then the total number of sequences of rules of length $2n - 1$ is given by m^{2n-1} , **which is an exponential function in the length n of an input string and becomes enormously large as n becomes large. Hence this way of solving the membership problem cannot be practical.***

We shall describe the CYK (Cocke-Younger-Kasami) algorithm which efficiently solves the membership problem, based on dynamic programming. In $O(n^3)$ time, the algorithm constructs a triangular table that tells whether w is in L .

In the CYK algorithm we construct a triangular table. The horizontal axis corresponds to the positions of the string $w = a_1 a_2 \cdots a_n$, which we have supposed has length 5.

The table entry X_{ij} is the set of variables A such that

$$A \Rightarrow^* a_i a_{i+1} \cdots a_j.$$

Note in particular that we are interested in whether S is in the set X_{1n} because that is the same as saying $S \Rightarrow^* w$, i.e., w is in L .

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

To fill the table we work **row by row upwards**. Notice that each row corresponds to one length of substrings; the bottom row is for strings of length 1, the second-from-bottom row for strings of length 2, and so on, until the top row corresponds to the one substring of length n , which is w itself.

It takes $O(n)$ time to compute any one entry of the table, by a method we shall discuss next. Since there are $n(n+1)/2$ **table entries**, the whole table construction process takes $O(n^3)$ time.

Since **Chomsky normal form** is assumed, $A \Rightarrow^* a_i a_{i+1} \cdots a_j$ holds if $a_i a_{i+1} \cdots a_j$ is divided into two substrings $a_i a_{i+1} \cdots a_k$ and $a_{k+1} a_{k+2} \cdots a_j$ in such a way that $B \Rightarrow^* a_i a_{i+1} \cdots a_k$ and $C \Rightarrow^* a_{k+1} a_{k+2} \cdots a_j$ holds for some nonterminals B, C and for some rule $A \rightarrow BC$.

In other words, to decide whether or not $A \in X_{ij}$, we only have to check whether or not there exists an integer k such that $i \leq k \leq j - 1$ and nonterminals B and C such that $B \in X_{ik}$, $C \in X_{k+1,j}$, and $A \rightarrow BC \in R$, where R denotes the set of rules.

6	X_{16}					
5	X_{15}	X_{26}				
4	X_{14}	X_{25}	X_{36}			
3	X_{13}	X_{24}	X_{35}	X_{46}		
2	X_{12}	X_{23}	X_{34}	X_{45}	X_{56}	
1	X_{11}	X_{22}	X_{33}	X_{44}	X_{55}	X_{66}
Row No./String	a_1	a_2	a_3	a_4	a_5	a_6

👉 The variables X_{11}, \dots, X_{nn} in row 1 of the table contain nonterminals that generate a_1, \dots, a_n , respectively.

👉 The variables in $X_{12}, \dots, X_{n-1,n}$ in row 2 contain nonterminals that generate $a_1a_2, a_2a_3, \dots, a_{n-1}a_n$, respectively.

👉 Similarly, nonterminals in rows numbered $3, \dots, n$ are specified.

👉 The algorithm to solve the membership problem somehow computes the nonterminals of the variables **from the bottom row toward the top row** and decides that $S \Rightarrow^* a_1a_2 \cdots a_n$ holds **if the variable X_{1n} contains the start symbol S .**

The problem left is **how to compute the nonterminals of each variable X_{ij}** , which will now be explained. First, let element X_{ii} be a set of all A such that $A \rightarrow a_i \in R$. Suppose that nonterminals in the variables that are placed in row 1 up to row $l - 1$ have been computed so far. **We shall describe how to compute nonterminals in X_{ij} 's that are placed on row l , where $n \geq l = j - i + 1 \geq 2$.**

☞ Clearly we have $A \Rightarrow^* a_i a_{i+1} \cdots a_j$ if there exists an integer k such that $i \leq k \leq j - 1$ and nonterminals B and C such that

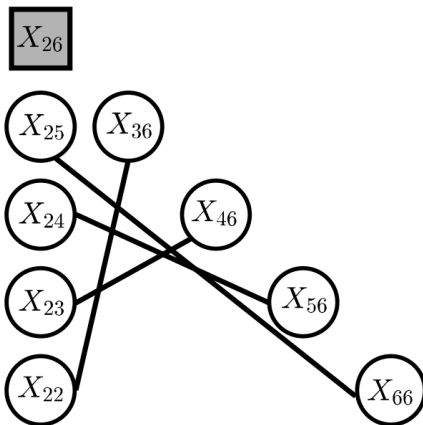
$$(1) B \in X_{ik}$$

$$(2) C \in X_{k+1,j}$$

$$(3) A \rightarrow BC \in R$$

☞ Note that, since $k - i + 1 \leq j - 1 - i + 1 = j - i < l$ and $j - (k + 1) + 1 \leq j - (i + 1) + 1 = j - i < l$, both X_{ik} and $X_{k+1,j}$ are placed in row 1, ..., and row $l - 1$. **Since nonterminals in X_{ik} and those in $X_{k+1,j}$ are already computed, we can check the conditions (1), (2), and (3).**

How to compute X_{26} from the four pairs (X_{22}, X_{36}) , (X_{23}, X_{46}) , (X_{24}, X_{56}) , and (X_{25}, X_{66}) ?



روند الگوریتم CYK

1. *Accept* if $a_1 a_2 \cdots a_n = \varepsilon$ and $S \rightarrow \varepsilon$ is a rule.
2. (computing the variables in the first row)
For each rule of type $A \rightarrow a$.
For each i such that $a = a_i$.
Add A to X_{ii} .
3. (computing the variables in the l th row where $l \geq 2$)
Repeat the following for $l = 2, \dots, n$.
Repeat the following for each X_{ij} in the l th row such that $j - i + 1 = l$.
Repeat the following for each rule $A \rightarrow BC$ and each $i \leq k \leq j - 1$.
If $B \in X_{ik}$ and $C \in X_{k+1,j}$, then add A to X_{ij} .
4. *Accept* if $S \in X_{1n}$.

The above-described algorithm is of order $O(n^3)$, which is a great improvement over the number of steps, given by $O(m^{2n-1})$, of the algorithm based on brute-force search described at the beginning of this part.

Example 5: We try to apply the algorithm described above to a context-free grammar given as follows:

$$\begin{aligned} S &\rightarrow AB|AC|AA, \\ A &\rightarrow CB|a, \\ B &\rightarrow AC|b, \\ C &\rightarrow CC|b. \end{aligned}$$

5	{S, A, B}				
4	{S, A}	{S, A, B}			
3	{S}	{A}	{S, B}		
2	{A, C}	\emptyset	{S, B}	{A, C}	
1	{B, C}	{B, C}	{A}	{B, C}	{B, C}
Row No./Column No.	1	2	3	4	5
String	b	b	a	b	b

For example, let us see how X_{25} is computed. Variable X_{25} comes from three pairs (X_{22}, X_{35}) , (X_{23}, X_{45}) , and (X_{24}, X_{55}) , which correspond to $(\{B, C\}, \{S, B\})$, $(\emptyset, \{A, C\})$, and $(\{A\}, \{B, C\})$, respectively. Since we have

$$\{B, C\} \cdot \{S, B\} = \{BS, BB, CS, CB\},$$

$$\emptyset \cdot \{A, C\} = \emptyset,$$

$$\{A\} \cdot \{B, C\} = \{AB, AC\},$$

and the rules such that these strings of length 2 appear on the right-hand side of the rules are

$$S \rightarrow AB, S \rightarrow AC, A \rightarrow CB \text{ and } B \rightarrow AC,$$

we have $X_{25} = \{S, A, B\}$. Other elements are similarly obtained. Since $X_{15} = \{S, A, B\}$, which includes the start symbol S , it is concluded that the string $bbabb$ is generated by the context-free grammar.

Example 6: The following are the productions of a CNF grammar G:

$$S \rightarrow AB|BC$$

$$A \rightarrow BA|a$$

$$B \rightarrow CC|b$$

$$C \rightarrow AB|a$$

We shall test for membership in $L(G)$ the string *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

👉 In order for a variable to generate ba , it must have a body whose first variable is in $X_{11} = \{B\}$ (i.e., it generates the b) and whose second variable is in $X_{22} = \{A, C\}$ (i.e., it generates the a). This body can only be BA or BC . If we inspect the grammar, we find that the productions $A \rightarrow BA$ and $S \rightarrow BC$ are the only ones with these bodies. Thus the two heads A and S constitute X_{12} .

👉 For a more complex example consider the computation of X_{24} . We can break the string aab that occupies positions 2 through 4 by ending the first string after position 2 or position 3. That is we may choose $k = 2$ or $k = 3$ in the definition of X_{24} . Thus we must consider all bodies in (X_{22}, X_{34}) and (X_{23}, X_{44}) .

$$\{A, C\} \cdot \{S, C\} = \{AS, AC, CS, CC\}$$

$$\{B\} \cdot \{B\} = \{BB\}$$

Of the five strings in this set only CC is a body, and its head is B . Thus $X_{24} = \{B\}$.

تمرین:

$$G: S \rightarrow aA \mid BD$$

$$A \rightarrow aA \mid aAB \mid aD$$

$$B \rightarrow aB \mid aC \mid BF$$

$$C \rightarrow Bb \mid aAC \mid E$$

$$D \rightarrow bD \mid bC \mid b$$

$$E \rightarrow aB \mid bC$$

$$F \rightarrow aF \mid aG \mid a$$

$$G \rightarrow a \mid b$$