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) □

CFL & intersection (we provide a counter example)

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

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

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

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

$L_1 \cap L_2 = \{0^n 1^n 2^n \mid n \geqslant 1\}$ 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 define

cartesian product

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

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

PDA

REG

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

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

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

if and only if

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

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, $w \in L \cap R$ □

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 $\setminus$ *with CFL* : Σ^* is a CFL. If $L_1 \setminus L_2$ would always be a CFL, then $\Sigma^* \setminus L = \overline{L}$ 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
  ↳ Consider L non CFL, $L \cap \Sigma^* = L$ is non CFL
- the intersection of a non-CFL and a finite language is always a CFL
  ↳ The intersection is always finite ⇒ regular ⇒ 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 → exponential time

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 **iteratively** 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 $X_{ii} \leftarrow \{A \mid (A \rightarrow a_i) \in P\}$

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

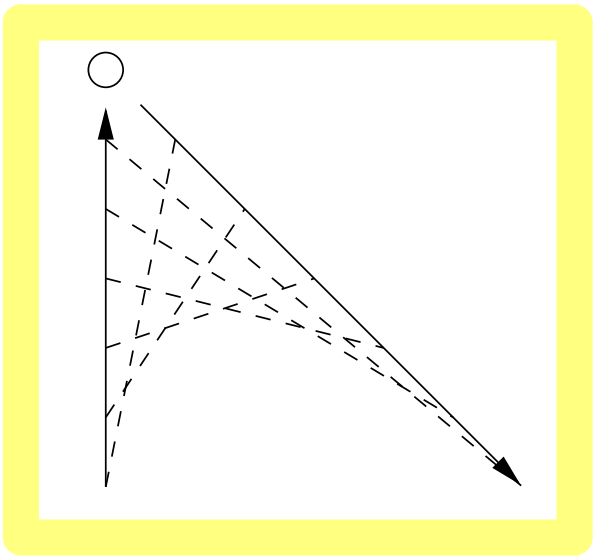
$X_{ij} \leftarrow X_{ij} \cup \{A\}$ if and only if there exist k, B, C such that

- $i \leqslant k < j$
- $(A \rightarrow BC) \in P$
- $B \in X_{ik}$ 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

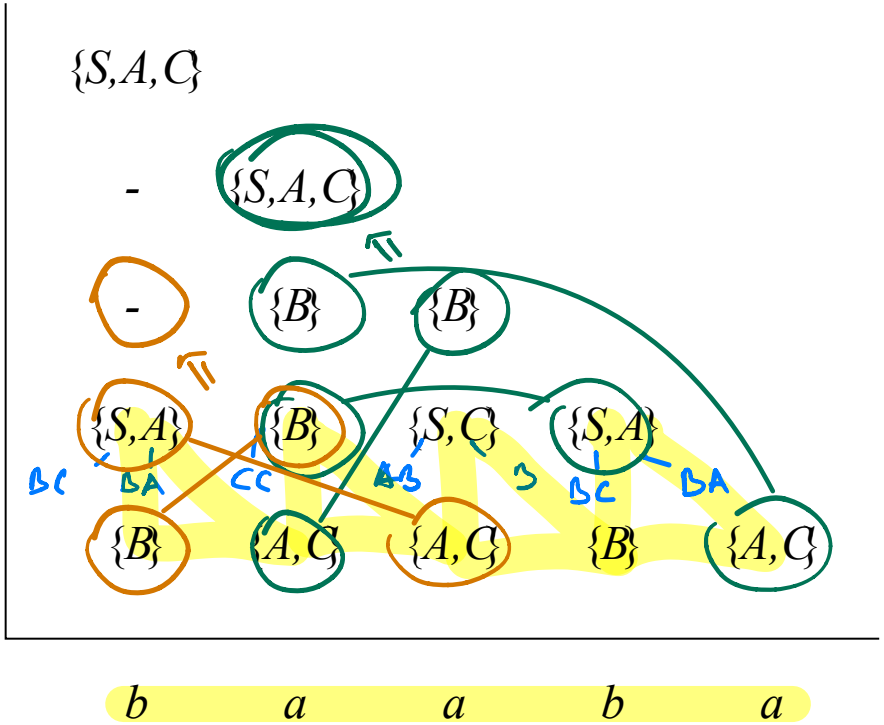
Theorem The algorithm for the construction of the parse table computes all of the sets X_{ij} in time $\mathcal{O}(n^3)$. We then have $w \in L(G)$ if and only if $S \in X_{1n}$

Example

Let G be a CFG with productions

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

and let $w = baaba$



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 $L = \Sigma^*$