13. Decision Algorithms for Context-free Languages

<u>Goal</u>: • Study algorithmic problems for context-free languages
• Focus on positive results, problems that are decidable (that can be solved algorithmically).

13.1 Emptiness and Inclusion in a Regular Language

<u>Goal</u>: Develop an algorithm that checks whether a given CFL is empty.

Formally, we study the following problem:

<u>EMPTYCFL</u>:

<u>Given</u>: A CFG $G = (N, \Sigma, P, S)$

<u>Question</u>: Is $L(G) = \emptyset$?

<u>Theorem</u>: EMPTYCFL is decidable in $O(|P|^2)$.

<u>Proof</u>:

- We compute an ascending chain of sets of non-terminals
  $$N_0 \subseteq N_1 \subseteq \ldots$$
  until we reach a fixed point $N_k = N_{k+1} = \bigcup_{i \in \mathbb{N}} N_i$.
- The idea is that N_i contains the non-terminals from which we can derive a terminal word with a parse tree of height $i+1$.
- Formally:
  $$N_0 := \{ A \in N \mid A \to w \in P \text{ with } w \in \Sigma^* \}$$
  $$N_{i+1} := \{ A \in N \mid A \to \alpha \in (N_i \cup \Sigma)^* \}.$$
- Like for finite automata, we only need to apply each production (at most) once.
- We still have to go through the remaining productions to find the applicable ones. $\square$

- In program verification, we study the problem whether each word in a context-free language (modeling a recursive program) is correct wrt. a (safety) specification.
- The specification is often given as a regular language, so the (safety) verification problem amounts to:

<u>INCLUSIONCFLREG</u>:

<u>Given</u>: A CFG G in CNF and an NFA A.

<u>Question</u>: Does $L(G) \subseteq L(A)$ hold?

<u>Theorem</u>: INCLUSIONCFLREG is decidable in $O(|G|^6 \cdot 2^{6|A|})$.

<u>Proof</u>: - The following equivalence is of great importance in verification:

$$\boxed{L(G) \subseteq L(A) \quad \text{iff} \quad L(G) \cap \overline{L(A)} = \emptyset}$$

- We can thus determinize A and invert the final states to obtain B with

$$L(B) = \overline{L(A)}.$$

This takes at most exponential time (for the powerset construction).

- The context-free languages are closed under regular intersection. We do the triple construction and obtain H with

$$L(H) = L(G) \cap L(B).$$

The triple construction introduces

$|N| \cdot (2^{|Q|})^2$ non-terminals and

$|\Sigma| \cdot |N| \cdot (2^{|Q|})^2$ productions $(Q_1, A, Q_2) \to a$ (for $A \to a$) and

$|N|^3 (2^{|Q|})^3$ productions $(Q_1, A, Q_2) \to (Q_1, B, Q)(Q, C, Q_2)$ (for $A \to BC$).

Checking emptiness works in quadratic time.

Altogether, the construction of A works in time

$$O(|G|^3 2^{3|A|}).$$

This upper bound is in particular due to the number of productions.
So we do not save by considering them separately.
Applying the emptiness check yields

$$O((|G|^3 \cdot 2^{3|A|})^2) = O(|G|^6 \cdot 2^{6|A|}). \qquad \square$$

Interestingly, the reverse inclusion

$$L(A) \subseteq L(G)$$

will turn out to be undecidable, even for a fixed language A.

<u>UNIVERSALITY CFL</u>:

<u>Given</u>: A CFG G over Σ.
<u>Question</u>: Is $L(G) = \Sigma^*$?

<u>Theorem</u>: UNIVERSALITYCFL is undecidable.
We will see the proof in later chapters.
As a consequence, checking whether a given CFL is regular
has to be undecidable.

<u>REGULARITYCFL</u>:

<u>Given</u>: A CFG G over Σ.
<u>Question</u>: Is L(G) regular and, if so, give an NFA for it.

<u>Theorem</u>: REGULARITYCFL is undecidable.
In later chapters we will see that the theorem
even holds without the "if so" requirement.
<u>Proof</u>: · Towards a contradiction, assume REGULARITYCFL was decidable.
· Using this assumption, we can construct an algorithm to solve UNIVERSALITYCFL ↯

This is a contradiction, there is no algorithm to solve universality.
Hence, there cannot be an algorithm for regularity.

- Let G be the input to the universality problem.
  We use the algorithm for REGULARITYCFL to check whether $L(G)$ is regular.
  If not, $L(G)$ cannot be Σ^* (because Σ^* is regular) and we <u>return false.</u>
  If so, REGULARITYCFL returns an NFA A for $L(G)$.
  We use A to check $L(A) = \Sigma^*$, and <u>return the answer.</u>

- Since this method solves UNIVERSALITYCFL, the assumption that REGULARITYCFL is decidable has to be false. □

<u>13.2 Membership and Dynamic Programming</u>

<u>Goal</u>:
- Show that membership is decidable (in polynomial time) for context-free languages.
- Introduce the algorithmic technique of <u>dynamic programming</u>

<u>Dynamic programming</u>:
- Accumulate information about smaller subproblems to solve larger problems
- Store solution to subproblems to avoid recomputing them (make a table where they are stored). <u>(memoization)</u>

<u>Example</u>: Fibonacci

Naive algorithm:
$$\begin{aligned} fib(5) &= fib(4) + fib(3) \\ &= (fib(3) + fib(2)) + (fib(2) + fib(1)) \\ &= (fib(2) + fib(1)) + \underline{fib(2)} + (\underline{fib(2)} + fib(1)) \\ &= (\underline{fib(2)} + \ldots \end{aligned}$$

Dynamic programming algorithm: Store $mem(0) := 0$, $mem(1) := 1$ and set $mem(n) := mem(n-1) + mem(n-2)$.

<u>Idea</u>: Memoization, and dynamic programming in general, is like computing a fixed point on auxiliary information.

<u>Definition</u>:
The problem MEMBERSHIP $L(G)$ with G a context-free grammar in Chomsky normal form (over Σ) is the membership problem for the language:

<u>Given</u>: Input word $w \in \Sigma^*$.
<u>Question</u>: Does $w \in L(G)$ hold?

<u>Idea for dynamic programming</u>: The subproblems determine for each non-terminal A of G and for every infix v of w whether $A \Rightarrow^* v$.

<u>Table</u>: The algorithm enters the solution into an $n \times n$ table, $n = |w|$.
For $i \leq j$, we have
$table(i,j) :=$ Non-terminals that generate $w_i \dots w_j$.
For $i > j$, the table entries are not used.

<u>Filling</u>:
- Fill the table entries for all infixes of w
- Increase in length: $\hookrightarrow$ Start from infixes of length 1
  $\hookrightarrow$ Continue with infixes of length 2
  $\hookrightarrow$ ...
- Key: Use entries for the shorter lengths to determine the entries for the longer lengths.

Accept: If start symbol S is in the set table$(1,n)$.

Details on filling: • Assume we have already determined which non-terminals generate all substrings of length $\leq k$.

• To determine whether A generates w of length $k+1$,

$$w = a_1 \dots a_{k+1},$$

split w into two non-empty pieces.
There are k possible ways of splitting w.

• For each split position m,
let $v_1 := a_1 \dots a_m$ and $v_2 := a_{m+1} \dots a_{k+1}$.
We examine all rules

$$A \to BC$$

and check whether

B generates v_1 and

C generates v_2.

If so, we add A to the entry for w.

Example: Consider

$$G = \begin{array}{l} S \to AB \mid BC \\ A \to BA \mid a \\ B \to CC \mid b \\ C \to AB \mid a \end{array} \quad \text{and} \quad w = baaba$$

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \{B\} & \{A,S\} & \emptyset & \emptyset & \{A,S,C\} \\ 2 & & \{A,C\} & \{B\} & \{B\} & \{A,C,S\} \\ 3 & & & \{A,C\} & \{S,C\} & \{B\} \\ 4 & & & & \{B\} & \{A,S\} \\ 5 & & & & & \{A,C\} \end{array}$$

We have $baaba \in L(G)$, because $S \in \{A,S,C\} = $ table$(1,5)$.

This dynamic programming algorithm is named after Cocke, Younger, and Kasami who presented it (independently) in the 1960s.

<u>CYK-Algorithm</u>: Let $G = (N, \Sigma, P, S)$.

<u>input</u>: Word $w = a_1 \dots a_n$.

<u>begin</u>:

<u>for all</u> $i = 1, \dots, n$ <u>do</u> // Initialize diagonal

table$(i,i) := \{A \in N \mid A \to a_i \in P\}$

<u>od</u>

<u>for all</u> $k = 2, \dots, n$ <u>do</u> // Length

<u>for all</u> $i = 1, \dots, (n-k)+1$ <u>do</u> // Start of the infix

table$(i, (i+k)-1) := \emptyset$ // Initialize table$(i, (i+k)-1)$

<u>for all</u> $m = 1, \dots, k-1$ <u>do</u> // Split length

table$(i, (i+k)-1) :=$ table$(i, (i+k)-1) \cup$

$\{A \in N \mid A \to BC$ with $B \in$ table$(i, (i+m)-1)$

and $C \in$ table$((i+m), (i+k)-1)\}$.

<u>end for all</u>

<u>end for all</u>

<u>end for all</u>

<u>return</u>: true, if $S \in$ table$(1,n)$

false, otherwise.

<u>end</u>.

<u>Complexity analysis</u>: • There are three nested loops

↳ Length of the infix

↳ Start position of the infix

↳ Split position.

• Hence, the runtime is $\mathcal{O}(|w|^3)$.

Theorem:

For every context-free grammar G,

MEMBERSHIP $L(G)$ can be solved in $O(|w|^3)$.

Note: The grammar is not part of the input to the problem.

Therefore, going through the rules $A \to BC$

only adds constant overhead.

13.3 Finiteness

Goal: Check whether a given CFL contains finitely many words.

Motivation: Such boundedness problems are also of importance in verification.

To burn a C-program into hardware,

we have to check that

↳ the stack is bounded in height and

↳ that it allocates a bounded amount of memory.

Note: This requires techniques different from the ones for emptiness.

We have to check that a loop can be repeated,

and hence need a kind of pumping argument.

FINITECFL:

Given: A CFG G.

Question: Is $L(G)$ finite?

Theorem: FINITECFL is decidable.

Proof: An inefficient algorithm can be derived from the pumping lemma.

We convert G into a grammar G' in Chomsky normal form.

Let k be the number of non-terminals in G'.

Let $p_L := 2^k$.

We check whether $L(G)$ contains a word w of length

$$p_L \leq |w| \leq 2p_L.$$

- If so, we return <u>false</u>, meaning the language is infinite. Clearly, w meets the conditions of the pumping lemma.
- If not, we return <u>true</u>, meaning the language is finite. Indeed, let u be the shortest word in $L(G)$ with $|u| \geq p_L$.
  We claim that $|u| \leq 2p_L$.
  Towards a contradiction, assume $|u| > 2p_L$.
  By the pumping lemma, $u = x_1 x_2 x_3 x_4 x_5$
  with $|x_2 x_3 x_4| \leq p_L$ and $x_1 x_3 x_5 \in L$.
  Since $|x_1 x_2 x_3 x_4 x_5| > 2p_L$ and $|x_2 x_3 x_4| \leq p_L$,
  we get $|x_1 x_3 x_5| \geq p_L$.
  A contradiction to minimality of u.
- We can solve these finitely many queries using CYK. $\square$

For a better algorithm, we turn G into a <u>CNF</u> G' for $L(G) \setminus \{\varepsilon\}$ <u>without</u> <u>useless</u> <u>non-terminals</u>.
We have $L(G)$ finite iff $L(G')$ is finite.
Let $G' = (N, \Sigma, P, S)$.
From G' we construct a directed graph (V, E) with

$V := N$ // Every non-terminal yields a node

$E := \{A \to B \mid A \to BC \in P \text{ or } A \to CB \in P\}.$

Claim: $L(G')$ is finite iff (V, E) is acyclic.

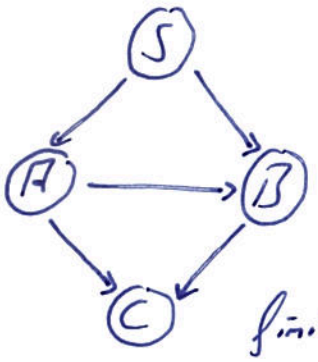
This holds since the non-terminals produced in a loop

- are guaranteed to derive a word (because they are not useless)
- and the word is not ε (because we have CNF).

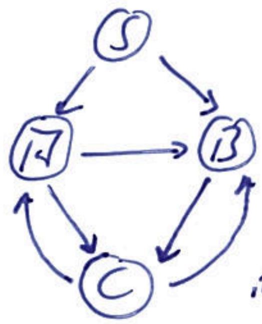
Example:

(1)
$S \to AB$
$A \to BC \mid a$
$B \to CC \mid b$
$C \to a$

(2)
$S \to AB$
$A \to BC \mid a$
$B \to CC \mid b$
$C \to AB$



finite.



infinite.