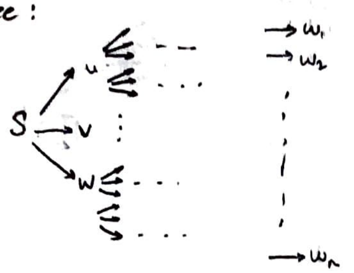
Edin Guso
23435

# CS407 Homework 5

Q1) Showing whether $L(G_1)$ and $L(G_2)$ are disjoint $\Rightarrow$ showing whether $L(G_1) \cap L(G_2) = \emptyset$

Since both $G_1$ and $G_2$ are CFGs, we can show all their possible production like a tree:



$w_i \; \forall 1 \le i \le n$ are the possible substrings of the resulting string. That is because $u, v, w, \ldots \in (V \cup T)^*$ and for each non-terminal there will be a new substring path.

Say, the resulting substrings of $G_1$ are $w_i$ and of $G_2$ are $x_i$. Now, since these $w_i$ and $x_i$ are ordered such that the leftmost substrings are $w_1$ and $x_1$, and rightmost substrings are $w_n$ and $x_n$, what we need to do to show the sets are disjoint (or equivalently if there is a common string) is to find out if any two ordered substring combinations are equal.

So, what we want to find is whether there is a sequence of integers

$1 \le i_1, i_2, \ldots, i_m \le n$ and $i_1 < i_2 < \ldots < i_m$ which gives us

$w_{i_1} w_{i_2} \ldots w_{i_m} = x_{i_1} x_{i_2} \ldots x_{i_m}$.

This would be equivalent to PCP if there wasn't the condition $i_1 < i_2 < \ldots < i_m$. However, when this condition is satisfied, it will behave exactly like PCP and thus will be undecidable. When this condition is not satisfied (i.e $\exists j,k$ s.t. $i_j > i_k$ & $j < k$), then this is not a valid production and is not in either of $L(G_1)$ or $L(G_2)$, thus it does not affect our solution and can be ignored.

Q2) Theorem: A language $L$ is generated by a grammar $G = (V, T, P, S) \Leftrightarrow$ there is a TM $M$ that semidecides $L$. This has been proven in lecture notes. Using this theorem, we can say that showing whether some $w \in L(G)$ is equivalent to showing whether some $w \in L(M)$. This is equivalent to asking whether $M$ halts on some $w$, which is the halting problem and is undecidable. Therefore, the initial problem is also undecidable.

Edin Guso
23435

4.6.3) Any production that consumes a single non-terminal can be written in the form $uAv \to uwv$ where $u$ = LHS of the consumed nonterminal, $v$ = RHS of the consumed nonterminal, $A$ = consumed nonterminal, and $w$ = generated symbols. This is trivial.

We need to convert all other productions (ones that consume $\geq 2$ nonterminals). Let's solve this for the general case where the number of nonterminals consumed is $n$:

$uA_1 \dots A_n v \to uwv$

We can decompose this single production into following productions:

$\underbrace{u}_{u}\underbrace{A_1}_{A}\underbrace{\dots A_n v}_{v} \to \underbrace{u}_{u}\underbrace{A_2 \dots A_n v}_{v}$, $\underbrace{u}_{u}\underbrace{A_2}_{A}\underbrace{\dots A_n v}_{v} \to \underbrace{u}_{u}\underbrace{A_3 \dots A_n v}_{v}$, ..., $\underbrace{u}_{u}\underbrace{A_n}_{A}\underbrace{v}_{v} \to \underbrace{u}_{u}\underbrace{w}_{w}\underbrace{v}_{v}$

and $w = e$ and $w = e$

This way, we can convert any grammar into an equivalent grammar with rules of the form $uAv \to uwv$ where $A \in V - \Sigma$, and $u, v, w \in V^*$.

5.7.4.)

a) Consider the set of strings with length $\leq |w|$ where $w \in V^*$ as the vertices in a directed graph where there is an edge from $x$ to $y$ iff $x \to y$. This is a finite graph, so we can compute reachability and determine all possible strings that can be produced of length $\leq |w|$. It is enough to consider strings of length $\leq |w|$ since if $|x| > |y|$ then no series of application of rules to $x$ can ever result in a string of length $\leq |y|$.

b) $\Leftarrow$ Since we have $w \neq e$, $|uAv| \leq |uwv|$. This is the necessary condition for a language to be context-sensitive.

$\Rightarrow$ Say we have a context-sensitive grammar with $r$ rules, each of the form $u \to v$ where $|u| \leq |v|$. Consider the $m$-th rule:

$\sigma_1 \sigma_2 \dots \sigma_n \to \tau_1 \tau_2 \dots \tau_{n+k}$, $k \geq 0$, $n \geq 1$

Introduce new nonterminals $A_1^m, \dots, A_n^m, B_1^m, \dots, B_k^m$ and replace the rule with the following rules:

$\longrightarrow$ Next Page

Edin Gusa
23435

- $A_1^m \dots A_{i-1}^m \sigma_i \sigma_{i+1} \dots \sigma_n \rightarrow A_1^m \dots A_{i-1}^m A_i^m \sigma_{i+1} \dots \sigma_n$ for each $1 \leq i \leq n$
- $\tau_1 \dots \tau_{i-1} A_i^m A_{i+1}^m \dots A_n^m \rightarrow \tau_1 \dots \tau_{i-1} \tau_i A_{i+1}^m \dots A_n^m$ for each $1 \leq i \leq n$
- $\tau_1 \dots \tau_{n-1} A_n^m \rightarrow \tau_1 \dots \tau_{n-1} \tau_n B_1$
- $B_i^m \rightarrow \tau_i B_{i+1}^m$ for each $1 \leq i < k$
- $B_k^m \rightarrow \tau_k$.

Each rule is now in the correct form. We also ensure that each $A$ is a nonterminal by changing every terminal in the original language into a nonterminal and adding new rules of the form $A \rightarrow a$ for each replaced terminal.

c) $\Rightarrow$ Assume $G$ is a context-sensitive language. Let $M$ be a NDTM that has all the rules of $G$ in reverse as its transition function. For every $u \rightarrow v$ in $G$, there will be $\delta(q, v) = (q', u)$ in $M$.

$M$ will work in a loop and each loop will start at the configuration $(q, \triangleright \# w)$. If $w = S$, then $M$ will halt. Otherwise, it will nondeterministically pick a rule to unapply and it will nondeterministically pick a position where to unapply this rule. If the rule can be unapplied at that position, then proceed. Otherwise enter an infinite loop. After unapplying the rule, if $|u| < |v|$, then we fill the created blanks with $\$ \notin \Sigma$, $\$$ will be ignored throughout the computation steps.

$\Leftarrow$ Assume $M$ is an in-place acceptor. A similar argument as above can be made, just in the reverse direction.

5.7.5)

1) Regular Languages: Accepted by finite automata
2) Context-free Languages: Accepted by pushdown automata
3) Context-sensitive Languages: Accepted by linear-bounded automata
4) Recursive Languages: Decided by Turing machines
5) Recursively Enumerable Languages: Accepted by unrestricted grammars