

Theory of Computation

Thomas Zeugmann

Hokkaido University
Laboratory for Algorithmics

<http://www-alg.ist.hokudai.ac.jp/~thomas/ToC/>

Lecture 10: \mathcal{CF} , PDAs and Beyond



Greibach Normal Form I

We want to show that all context-free languages are accepted by pushdown automata. For doing this, it is very convenient to use another normal form for context-free languages, i.e., the so-called Greibach normal form.

Greibach Normal Form I

We want to show that all context-free languages are accepted by pushdown automata. For doing this, it is very convenient to use another normal form for context-free languages, i.e., the so-called Greibach normal form.

Definition 1

A context-free grammar $\mathcal{G} = [T, N, \sigma, P]$ with $\lambda \notin L(\mathcal{G})$ is said to be in *Greibach normal form* if every production of \mathcal{G} has the form $h \rightarrow a\alpha$, where $a \in T$ and $\alpha \in N^*$.

Greibach Normal Form II

Clearly, we aim to show that every context-free language does possess a grammar in Greibach normal form. This is, however, not as easy as it might seem. Therefore, we skip the proof here and refer the audience to the book which also contains an example.

Greibach Normal Form II

Clearly, we aim to show that every context-free language does possess a grammar in Greibach normal form. This is, however, not as easy as it might seem. Therefore, we skip the proof here and refer the audience to the book which also contains an example.

Theorem 1

For every language $L \in \mathcal{CF}$ with $\lambda \notin L$ there exists a grammar $\tilde{\mathcal{G}}$ such that $L = L(\tilde{\mathcal{G}})$ and $\tilde{\mathcal{G}}$ is in Greibach normal form.

From \mathcal{CF} to Pushdown Automata I

Now, we are ready to show the remaining fundamental theorem concerning the power of pushdown automata.

From \mathcal{CF} to Pushdown Automata I

Now, we are ready to show the remaining fundamental theorem concerning the power of pushdown automata.

Theorem 2

For every language $L \in \mathcal{CF}$ there exists a pushdown automaton \mathcal{K} such that $L = N(\mathcal{K})$.

Proof. We assume that $\lambda \notin L$. It is left as an exercise to modify the construction for the case that $\lambda \in L$. Let $\mathcal{G} = [T, N, \sigma, P]$ be a context-free grammar in Greibach normal form such that $L = L(\mathcal{G})$.

From \mathcal{CF} to Pushdown Automata II

We need the following notation:

A string α of terminals and nonterminals is called a *sentential form* if $\sigma \xRightarrow{*} \alpha$.

form if $\sigma \xRightarrow{*} \alpha$.

Now, let

$$\mathcal{K} = [\{q\}, T, N, \delta, q, \sigma, \emptyset] ,$$

where $(q, \gamma) \in \delta(q, a, A)$ whenever $(A \rightarrow a\gamma) \in P$.

From $\mathcal{C}\mathcal{F}$ to Pushdown Automata II

We need the following notation:

A string α of terminals and nonterminals is called a *sentential form* if $\sigma \xRightarrow{*} \alpha$.

form if $\sigma \xRightarrow{*} \alpha$.

Now, let

$$\mathcal{K} = [\{q\}, T, N, \delta, q, \sigma, \emptyset] ,$$

where $(q, \gamma) \in \delta(q, a, A)$ whenever $(A \rightarrow a\gamma) \in P$.

The pushdown automaton \mathcal{K} simulates leftmost derivations of \mathcal{G} . Since \mathcal{G} is in Greibach normal form, each sentential form in a leftmost derivation consists of a string x of terminals followed by a string of nonterminals α .

From \mathcal{CF} to Pushdown Automata III

\mathcal{K} stores the suffix α of the left sentential form on its stack after processing the prefix x . Formally, we show the following claim.

Claim 1. $\sigma \xrightarrow[\mathcal{S}]{*} x\alpha$ *by a leftmost derivation if and only if*

$$(q, x, \sigma) \xrightarrow[\mathcal{K}]{*} (q, \lambda, \alpha).$$

From \mathcal{CF} to Pushdown Automata IV

We start with the sufficiency. The prove is done by induction.

That is, we assume $(q, x, \sigma) \xrightarrow[\mathcal{K}]{i} (q, \lambda, \alpha)$ and show $\sigma \xrightarrow[\mathcal{G}]{*} x\alpha$.

For the **induction basis**, let $i = 0$. That is, we assume

$(q, x, \sigma) \xrightarrow[\mathcal{K}]{0} (q, \lambda, \alpha)$. By the definition of the reflexive transitive

closure $\xrightarrow[\mathcal{K}]{*}$, this means nothing else than $(q, x, \sigma) = (q, \lambda, \alpha)$.

Consequently, $x = \lambda$ and $\alpha = \sigma$. Obviously, by the definition of the reflexive transitive closure $\xrightarrow[\mathcal{G}]{*}$ we can conclude $\sigma \xrightarrow[\mathcal{G}]{*} \sigma$,

again in zero steps. This proves the induction basis.

From \mathcal{CF} to Pushdown Automata V

For the induction step, assume $i \geq 1$ and let $x = y\alpha$, where $y \in T^*$. Now, we consider the next-to-last-step, i.e.,

$$(q, y\alpha, \sigma) \xrightarrow[\mathcal{K}]{i-1} (q, \alpha, \beta) \xrightarrow[\mathcal{K}]{1} (q, \lambda, \alpha). \quad (1)$$

If we remove α from the end of the input string in the first i instantaneous descriptions of the sequence (1), we discover that

$$(q, y, \sigma) \xrightarrow[\mathcal{K}]{i-1} (q, \lambda, \beta), \text{ since } \alpha \text{ cannot influence } \mathcal{K}'\text{s behavior}$$

until it is actually consumed from the input. Thus, we can apply the induction hypothesis and obtain

$$\sigma \xrightarrow[\mathcal{G}]{*} y\beta.$$

From \mathcal{CF} to Pushdown Automata VI

Taking into account that the pushdown automaton \mathcal{K} , while

consuming a , is making the move $(q, a, \beta) \xrightarrow{1} (q, \lambda, \alpha)$, we

directly get by construction that $\beta = A\gamma$ for some $A \in N$,
 $(A \rightarrow a\eta) \in P$ and $\alpha = \eta\gamma$.

Hence, we arrive at

$$\sigma \xrightarrow[\mathcal{G}]{*} y\beta \xrightarrow[\mathcal{G}]{1} ya\eta\gamma = x\alpha.$$

This completes the **sufficiency** proof.

From \mathcal{CF} to Pushdown Automata VII

For showing the necessity suppose that $\sigma \xrightarrow[\mathcal{G}]{i} \lambda\alpha$ by a leftmost derivation. We prove by induction on i that

$(q, \lambda, \sigma) \xrightarrow[\mathcal{K}]{*} (q, \lambda, \alpha)$. The induction basis is again done for

$i = 0$, and can be shown by using similar arguments as above.

From \mathcal{CF} to Pushdown Automata VIII

For the induction step let $i \geq 1$ and suppose

$$\sigma \xrightarrow[\mathcal{G}]{i-1} yA\gamma \xrightarrow[\mathcal{G}]{1} y\alpha\eta\gamma, \text{ where } x = y\alpha, \alpha = \eta\gamma.$$

By the induction hypothesis we directly get

$$(q, y, \sigma) \xrightarrow[\mathcal{K}]{*} (q, \lambda, A\gamma)$$

and thus $(q, y\alpha, \sigma) \xrightarrow[\mathcal{K}]{*} (q, \alpha, A\gamma)$. Since $(A \rightarrow \alpha\eta) \in P$, we can

conclude that $(q, \eta) \in \delta(q, \alpha, A)$. Thus,

$$(q, x, \sigma) \xrightarrow[\mathcal{K}]{*} (q, \alpha, A\gamma) \xrightarrow[\mathcal{K}]{1} (q, \lambda, \alpha),$$

and the **necessity** follows. This proves Claim 1.

From \mathcal{CF} to Pushdown Automata IX

To conclude the proof of the theorem, we have only to note that Claim 1 with $\alpha = \lambda$ says

$$\sigma \underset{\mathcal{G}}{\overset{*}{\implies}} x \text{ if and only if } (q, x, \sigma) \underset{\mathcal{K}}{\overset{*}{\longrightarrow}} (q, \lambda, \lambda).$$

That is, $x \in L(\mathcal{G})$ if and only if $x \in N(\mathcal{K})$. █

Main Theorem

Now, putting it all together, we directly obtain the following.

Theorem 3

Let L be any language. Then the following three assertions are equivalent:

- (1) $L \in \mathcal{CF}$.
- (2) *There exists a pushdown automaton \mathcal{K}_1 such that $L = L(\mathcal{K}_1)$.*
- (3) *There exists a pushdown automaton \mathcal{K}_2 such that $L = N(\mathcal{K}_2)$.*

Remarks I

After so much progress we may want to ask questions like whether or not $L(\mathcal{G}_1) \cap L(\mathcal{G}_2) = \emptyset$ for any given context-free grammars $\mathcal{G}_1, \mathcal{G}_2$.

Remarks I

After so much progress we may want to ask questions like whether or not $L(\mathcal{G}_1) \cap L(\mathcal{G}_2) = \emptyset$ for any given context-free grammars $\mathcal{G}_1, \mathcal{G}_2$.

You may be tempted to think this is not a too difficult task. But as a matter of fact, nobody succeeded to design an algorithm solving this problem for all context-free grammars. Maybe, there is a deeper reason behind this situation. Before we can explore such problems, we have to deal with computability.

Remarks II

On the other hand, so far we have studied regular and context-free languages. But we have already seen a language which is not context-free, i.e.,

$$L = \{a^n b^n c^n \mid n \in \mathbb{N}\}.$$

Thus, it is only natural to ask what other language families are around. Due to the lack of time, we can only sketch these parts of formal language theory.

Context-Sensitive Languages I

First, we provide a formal definition.

Definition 2

A Grammar $\mathcal{G} = [T, N, \sigma, P]$ is said to be *context-sensitive* if all its productions satisfy the following conditions.

- (1) $(\alpha \rightarrow \beta) \in P$ iff there are s_1, s_2, r , and h such that $h \in N$, $s_1, s_2 \in (T \cup N)^*$ and $r \in (T \cup N)^+$ and $\alpha = s_1 h s_2$ and $\beta = s_1 r s_2$, or
- (2) $\alpha = h$ and $\beta = \lambda$ and h does not occur at any right-hand side of a production from P .

Context-Sensitive Languages II

Definition 3

A language is said to be *context-sensitive* if there exists a context-sensitive grammar \mathcal{G} such that $L = L(\mathcal{G})$.

Context-Sensitive Languages II

Definition 3

A language is said to be *context-sensitive* if there exists a context-sensitive grammar \mathcal{G} such that $L = L(\mathcal{G})$.

By \mathcal{CS} we denote the family of all context-sensitive languages. The name context-sensitive is quite intuitive, since the replacement or rewriting of a nonterminal is only possible in a certain context expressed by a prefix s_1 and suffix s_2 . The definition provided above directly allows for the observation that $\mathcal{CF} \subseteq \mathcal{CS}$.

Context-Sensitive Languages II

Definition 3

A language is said to be *context-sensitive* if there exists a context-sensitive grammar \mathcal{G} such that $L = L(\mathcal{G})$.

By \mathcal{CS} we denote the family of all context-sensitive languages. The name context-sensitive is quite intuitive, since the replacement or rewriting of a nonterminal is only possible in a certain context expressed by a prefix s_1 and suffix s_2 . The definition provided above directly allows for the observation that $\mathcal{CF} \subseteq \mathcal{CS}$.

Exercise: *Prove that $\mathcal{CF} \subset \mathcal{CS}$.*

Context-Sensitive Languages III

Using similar ideas as we did for \mathcal{REG} and \mathcal{CF} , one can show the following.

Theorem 4

The context-sensitive languages are closed under union, product and Kleene closure.

Proof. The proof is left as an exercise.

Context-Sensitive Languages III

Using similar ideas as we did for \mathcal{REG} and \mathcal{CF} , one can show the following.

Theorem 4

The context-sensitive languages are closed under union, product and Kleene closure.

Proof. The proof is left as an exercise.

Also, in the same way as Theorem 6.4 has been shown, one can prove the context-sensitive languages to be closed under transposition. That is, we directly get the next theorem.

Context-Sensitive Languages III

Using similar ideas as we did for \mathcal{REG} and \mathcal{CF} , one can show the following.

Theorem 4

The context-sensitive languages are closed under union, product and Kleene closure.

Proof. The proof is left as an exercise.

Also, in the same way as Theorem 6.4 has been shown, one can prove the context-sensitive languages to be closed under transposition. That is, we directly get the next theorem.

Theorem 5

Let Σ be any alphabet, and let $L \subseteq \Sigma^$. Then we have: If $L \in \mathcal{CS}$ then $L^T \in \mathcal{CS}$, too.*

Context-Sensitive Languages IV

Moreover, in contrast to the context-free languages we have:

Theorem 6

The context-sensitive languages are closed under intersection.

Context-Sensitive Languages IV

Moreover, in contrast to the context-free languages we have:

Theorem 6

The context-sensitive languages are closed under intersection.

For establishing further properties of context-sensitive languages, we need the following definition.

Definition 4

A Grammar $\mathcal{G} = [T, N, \sigma, P]$ is said to be **length-increasing** if for each production $(\alpha \rightarrow \beta) \in P$ the condition $|\alpha| \leq |\beta|$ is satisfied. In addition, P may contain the production $\sigma \rightarrow \lambda$ and in this case σ does not occur on the right-hand side of any production from P .

Context-Sensitive Languages V

Looking at the definition of context-sensitive grammars, we directly get the following corollary.

Corollary 7

Every context-sensitive grammar is length-increasing.

Context-Sensitive Languages V

Looking at the definition of context-sensitive grammars, we directly get the following corollary.

Corollary 7

Every context-sensitive grammar is length-increasing.

However, one can prove that there are length-increasing grammars which are not context-sensitive (see below).

Context-Sensitive Languages V

Looking at the definition of context-sensitive grammars, we directly get the following corollary.

Corollary 7

Every context-sensitive grammar is length-increasing.

However, one can prove that there are length-increasing grammars which are not context-sensitive (see below).

Nevertheless, one can show the following.

Theorem 8

For every length-increasing grammar there exists an equivalent context-sensitive grammar.

Context-Sensitive Languages VI

Putting it all together directly yields the following equivalence.

Theorem 9

Let L be any language. Then the following statements are equivalent:

- (1) There exists a context-sensitive grammar \mathcal{G} such that $L = L(\mathcal{G})$.*
- (2) There exists a length-increasing grammar $\tilde{\mathcal{G}}$ such that $L = L(\tilde{\mathcal{G}})$.*

Context-Sensitive Languages VII

Example 1

Let T be any alphabet. We define a grammar $\mathcal{G} = [T, N, \sigma, P]$ as follows. Let

$$N = \{\sigma\} \cup \{X_i \mid i \in T\} \cup \{A_i \mid i \in T\} \cup \{B_i \mid i \in T\}.$$

and let P be the following set of productions

- 1 $\sigma \rightarrow i\sigma X_i$
- 2 $\sigma \rightarrow A_i B_i$
- 3 $B_i X_j \rightarrow X_j B_i$
- 4 $B_i \rightarrow i$
- 5 $A_i X_j \rightarrow A_i B_j$
- 6 $A_i \rightarrow i$

where $i, j \in T$.

Context-Sensitive Languages VIII

Inspecting the productions we see that \mathcal{G} is a length-increasing grammar which is **not context-sensitive**, since the context in Production 3 is destroyed.

Nevertheless, by the latter theorem we know that the language $L(\mathcal{G})$ is context-sensitive.

Context-Sensitive Languages VIII

Inspecting the productions we see that \mathcal{G} is a length-increasing grammar which is **not context-sensitive**, since the context in Production 3 is destroyed.

Nevertheless, by the latter theorem we know that the language $L(\mathcal{G})$ is context-sensitive.

Exercise: *Prove that the grammar \mathcal{G} given in the example above generates*

$$L = \{ww \mid w \in T^+\}.$$

Context-Sensitive Languages IX

The notion of length-increasing grammar has another nice **implication** which we state next.

Theorem 10

There is an algorithm that on input any context-sensitive grammar $\mathcal{G} = [T, N, \sigma, P]$ and any string $s \in T^$ decides whether or not $s \in L(\mathcal{G})$.*

Context-Sensitive Languages X

Proof Sketch: Since every context-sensitive grammar is also a length-increasing grammar, it suffices to examine all finite sequences w_0, w_1, \dots, w_n with $|w_i| < |w_{i+1}|$, $i = 0, \dots, n - 1$ and $\sigma = w_0$ as well as $w_n = s$, where $w_i \in (T \cup N)^+$. The number of all those sequences is finite. Let \mathcal{S} be the set of all such sequences.

Context-Sensitive Languages X

Proof Sketch: Since every context-sensitive grammar is also a length-increasing grammar, it suffices to examine all finite sequences w_0, w_1, \dots, w_n with $|w_i| < |w_{i+1}|$, $i = 0, \dots, n - 1$ and $\sigma = w_0$ as well as $w_n = s$, where $w_i \in (T \cup N)^+$. The number of all those sequences is finite. Let \mathcal{S} be the set of all such sequences.

Now, the only thing one has to check is whether or not

$$w_i \xrightarrow[\mathcal{G}]{1} w_{i+1} \quad \text{for all } i = 0, \dots, n - 1. \quad (2)$$

Thus, one either finds a sequence in \mathcal{S} fulfilling (2). Then one can directly conclude that $s \in L(\mathcal{G})$. If all sequences in \mathcal{S} fail to satisfy (2) then $s \notin L(\mathcal{G})$. █

Context-Sensitive Languages XI

Recall that we have proved the regular languages to be closed under complement and the context-free languages to be *not* closed under complement. As curious as we are, we clearly like to know whether or not the context-sensitive languages are closed under complement.

Context-Sensitive Languages XI

Recall that we have proved the regular languages to be closed under complement and the context-free languages to be *not* closed under complement. As curious as we are, we clearly like to know whether or not the context-sensitive languages are closed under complement.

To answer this question is by no means easy. As a matter of fact, it took more than 20 years to resolve this question. So we have to be a bit patient. We shall provide an answer in the course on “Complexity and Cryptography.”

Beyond I

Finally, we mention what happens if we pose no restrictions whatsoever on the set of productions. We shall call the resulting family of languages \mathcal{L}_0 , where 0 should remind you to *zero* restrictions.

Beyond I

Finally, we mention what happens if we pose no restrictions whatsoever on the set of productions. We shall call the resulting family of languages \mathcal{L}_0 , where 0 should remind you to *zero* restrictions.

Then, it is quite obvious that all languages L in \mathcal{L}_0 share the property that we can algorithmically *enumerate all and only* the elements contained in L provided we are given a grammar \mathcal{G} for L . However, as we shall see later, our last **Theorem** *cannot* be generalized to \mathcal{L}_0 . So, we can directly conclude that $\mathcal{CS} \subset \mathcal{L}_0$.

Beyond II

Putting it all together gives us the famous **Chomsky Hierarchy**,
i.e.,

$$\mathcal{REG} \subset \mathcal{CF} \subset \mathcal{CS} \subset \mathcal{L}_0.$$

Thank you!