3515ICT: Theory of Computation

Context-free languages

Context-free grammars (H, Chapter 5; S, Section 2.1)

Example. Context-free grammar (CFG) for the language L_{pal} of palindromes of 0s and 1s (H, Fig. 5.1): $P \to \varepsilon \mid 0 \mid 1 \mid 0P0 \mid 1P1$. Note that L_{pal} is not regular.

Example. CFG for the language L_{bal} of balanced parenthesis strings: $B \to \varepsilon \mid BB \mid (B)$. Again, L_{bal} is not regular.

Definition of a CFG: $G = (V, \Sigma, R, S)$, variables (nonterminal symbols) V, terminal symbols Σ, productions R, start symbol S.

Example. CFG for the language of (arithmetic) expressions over identifiers (I) (H, Fig. 5.2): $E \to I \mid (E) \mid E + E \mid E * E$.

Derivation of sentences (terminal strings) from S. Distinction between $\to$, $\Rightarrow$, and $\overset{*}{\Rightarrow}$. Derivation of the expression $a * (a + b00)$ from E.

Leftmost and rightmost derivations: Replace the leftmost (resp., rightmost) variable at each step.

Every derivation of a sentence has equivalent leftmost and rightmost derivations.

Definition of the language $L(G)$ of a CFG G: the set of sentences (terminal strings) that have derivations from the start symbol of G:

$$L(G) = \{\, w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w \,\}$$

Definition of a context-free language (CFL): A language is a CFL if it is the language of some CFG.

Theorem. Every regular language is context-free. *Proof.* From the DFA for L, construct a CFG in which every production has one of the following forms:

$$\begin{aligned} &A \to \varepsilon \\ &A \to aB \end{aligned}$$

Theorem. A CFL is regular if and only if it has a CFG of the above (regular) form.

Note. Every CFL over $\Sigma = \{1\}$ is regular.

Not all languages are CFLs! The following languages are not context-free:

$$\begin{aligned} &\{\, a^n b^n c^n \mid n \geq 1 \,\} \\ &\{\, a^i b^j \mid j = i^2 \,\} \\ &\{\, ww \mid w \in \{a, b\}^* \,\} \end{aligned}$$

A sentential form α is a string of symbols in $\{V \cup \Sigma\}^*$ such that $S \overset{*}{\Rightarrow} \alpha$.

Example (H, Exercise 5.1.2): The following grammar generates the language of regular expression $0^*1(0+1)^*$.

$$
\begin{aligned}
&S \to A1B \\
&A \to 0A \mid \varepsilon \\
&B \to 0B \mid 1B \mid \varepsilon
\end{aligned}
$$

Note that this grammar is not regular, even though the language is.

Exercise. Construct a regular grammar for this language.

Parse trees (derivation trees) for a CFG correspond to derivations of sentential forms for the CFG. The yield of a parse tree is the concatenation of the leaves of the tree.

Theorem. Given a CFG $G = (V, \Sigma, R, S)$, the following are equivalent:

1. The recursive inference procedure determines that the terminal string w is in the language of variable A.
2. There is a derivation of w from A ($A \overset{*}{\Rightarrow} w$).
3. There is a leftmost (resp., rightmost) derivation of w from A.
4. There is a parse tree with root A and yield w.

Examples of transformations from trees to derivations and vice versa for the expression grammar.

Applications of CFGs (H, 5.3): programming language definition and implementation (parsing is the process of reconstructing a parse tree from a sentence), document-type definitions (DTDs) in XML.

Form of DTDs in XML (Ex. 5.14):

```
<!DOCTYPE PcSpecs [
    <!ELEMENT PCs (PC⋆)>
    <!ELEMENT PC (MODEL, PRICE, PROC, RAM, DISK+)>
    ...
    <!ELEMENT DISK (HARDDISK | CD | DVD)>
    <!ELEMENT HARDDISK (MANF, MODEL, SIZE)>
    <!ELEMENT MANF (\#PCDATA)>
    ...
]>
```

Note that here and in general, using regular expressions in the right-hand sides of productions gives no additional power: every such grammar may be transformed to an equivalent CFG.

Clearly, the class of CFLs is closed under union, concatenation, iteration and reversal. It is *not* closed under complementation, intersection or difference.

With the original grammar for expressions (H, Fig. 5.2), the sentential form $E + E * E$ has two derivations from E and two parse trees with root E. One derivation corresponds to the interpretation $E + (E * E)$ and the other to the interpretation $(E + E) * E$.

Definition. A CFG $G = (V, \Sigma, R, S)$ is ambiguous if there is at least one string w in Σ^* for which there are two different parse trees with root S and yield w.

Example of an unambiguous grammar for expressions (H, Fig. 5.19):

$$
\begin{aligned}
&E \to T \mid E + T \\
&T \to F \mid T * F \\
&F \to I \mid (E)
\end{aligned}
$$

Theorem. For every CFG $G = (V, \Sigma, R, S)$ and string w in Σ^*, w has two distinct parse trees with root S if and only if w has two distinct leftmost derivations from S.

Hence, for every unambiguous CFG $G = (V, \Sigma, R, S)$ and string w in $L(G)$, there exists a unique leftmost derivation of w from S.

Definition. A CFL L is inherently ambiguous if every grammar for L is ambiguous.

The language of expressions is not inherently ambiguous.

The following language is inherently ambiguous:

$$\{ a^i b^j c^k \mid i = j \text{ or } j = k \}$$

Intuitively, the reason is that there are always two distinct parse trees for the strings $a^n b^n c^n$ (for $n \geq 1$).

Pushdown automata (H, Chapter 6; S, Section 2.2)

Now, how do we define automata that recognise, and hence define, CFLs in general?

Informally, a pushdown automaton (PDA) is a nondeterministic finite-state automaton with empty transitions and a stack for storage, in which each transition depends on the state, input symbol and stack top, and has the effect of changing state and replacing the stack top by zero or more other symbols. The stack allows the automaton to remember indefinitely many symbols, but it can only use them in a restricted way.

(H, Example 6.1; S, Example 2.18) Consider a PDA for the language

$$L_{wwr} = \{ ww^R \mid w \in \{0, 1\}^* \}$$

In state q_0 we push a "stack bottom" symbol \$ onto the stack, and change to state q_1.

In state q_1, we repeatedly push symbols onto the stack, assuming we have not yet reached the middle of the string. At any time, we may "guess" that we have read the string w, and change to state q_2.

In state q_2, we attempt to recognise the string w^R by repeatedly matching the input symbol with the stack top, and pushing the stack top. If the two symbols don't match, this branch dies.

If the stack top is \$, we change to the final state q_3 and if we are at the end of the input, we accept the input string, which must have the form ww^R, and halt.

Definition of a PDA: A PDA P has a set of states Q, an input alphabet Σ, a stack alphabet Γ, a transition function δ, a start state $q_0 \in \Sigma$, and a set of final states $F \subseteq Q$. The transition function δ maps $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \cup \{\varepsilon\}$ into a subset of $Q \times \Gamma^*$.

For example, the PDA for L_{wwr} can be described as

$$P = (\{q_0, q_1, q_2, q_3\}, \{0,1\}.\{0,1,\$\}, \delta, q_0, \{q_3\})$$

where

1. $\delta(q_0, \epsilon, \epsilon) = \{(q_1, \$)\}$
2. $\delta(q_1, 0, \epsilon) = \{(q_1, 0)\}$ and $\delta(q_1, 1, \epsilon) = \{(q_1, 1)\}$
3. $\delta(q_1, \varepsilon, \varepsilon) = \{(q_2, \epsilon)$
4. $\delta(q_2, 0, 0) = \{(q_2, \varepsilon)\}$ and $\delta(q_2, 1, 1) = \{(q_2, \varepsilon)\}$
5. $\delta(q_2, \varepsilon, \$) = \{(q_3, \epsilon))\}$

Graphical notation for PDAs: an arc from p to q labelled $a, X \rightarrow \alpha$ means that $\delta(p, a, X)$ contains the pair (q, α), perhaps among other pairs.

An instantaneous description of a PDA is a triple (q, w, λ).

PDA moves are described as follows. Suppose (p, a, X) contains (q, α). Then for all strings $w \in \Sigma^*$ and $\beta \in \Gamma^*$:

$$(p, aw, X\beta) \vdash (q, w, \alpha\beta)$$

We use the symbol $\vdash^*$ to indicate a sequence of zero or more such moves.

A PDA may accept an input string either when it reaches a final state or when the stack becomes empty. We normally assume acceptance is by final state.

The language $L(P)$ accepted by a PDA P by final state is

$$\{\, w \mid (q_0, w, Z_0) \vdash^* (q, \varepsilon, \alpha) \,\}$$

for some state q in F and any stack string α.

Example. PDA for the language L_{bal} of balanced parenthesis strings.

The language $N(P)$ accepted by a PDA P by empty stack is

$$\{\, w \mid (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon) \,\}$$

Theorem. A language L has a PDA that accepts it by final state if and only if it has a PDA that accepts it by empty stack.

(If) Suppose P accepts L by empty stack. Use the construction of H, Fig. 6.4 to construct a PDA P' that accepts L by final state.

Example. Convert PDA by empty stack for "if-else" language to a PDA by final state (H, Example 6.10).

(Only if) Now suppose P accepts L by final state. Use the construction of H, Fig. 6.7 to construct a PDA P' that accepts L by empty stack.

Theorem. A language has a context-free grammar if and only if it is accepted by some PDA (by final state or by empty stack).

(Grammars to PDAs) Let $G = (V, \Sigma, R, S)$ be a CFG. Define a PDA P that accepts $L(G)$ by empty stack as follows:

$$P = (\{q\}, \Sigma, V \cup \Sigma, \delta, q, S)$$

where the transition function δ is defined by:

1. For each variable A, $\delta(q, \varepsilon, A) = \{ (q, \alpha) \mid A \to \alpha \in R \}$
2. For each terminal a, $\delta(q, a, a) = \{(q, \varepsilon)\}$

Now, for every leftmost derivation $S \overset{*}{\Rightarrow} w$ in G, there exists a corresponding sequence of moves $(q, w, S) \vdash^* (q, \varepsilon)$ in P. At each intermediate stage $xA\alpha$ in the derivation, there is a corresponding instantaneous description $(w, y, A\alpha)$ for P, where the input $w = xy$. Each move of the PDA either replaces a variable on the stack by one of its right hand sides, or reads a symbol from the input and pops that symbol from the stack.

Example. Convert the (simplified) expression grammar

$$E \to a \mid E * E \mid E + E \mid (E)$$

to the PDA with state $Q = \{q\}$, input symbols $\Sigma = \{a, *, +, (,)\}$, stack symbols $\Gamma = \Sigma \cup \{E\}$, and start symbol E. The transition function for the PDA is:

1. $(q, \varepsilon, E) = \{(q, a), (q, E * E), (q, E + E), (q, (E)).$
2. $(q, a, a) = (q, \varepsilon)$, $(q, *, *) = (q, \varepsilon)$, $(q, +, +) = (q, \varepsilon)$, $(q, (, () = (q, \varepsilon)$, $(q,),)) = (q, \varepsilon)$.

Note that this PDA is (very) nondeterministic.

Exercise. Convert the grammar for L_{pal} to a PDA.

Exercise. Convert the following LL(1) grammar for expressions to a PDA.

$$\begin{aligned} &E \to TX \\ &T \to FY \\ &X \to \varepsilon \mid +TX \\ &F \to a \mid (E) \\ &Y \to \varepsilon \mid *FY \end{aligned}$$

Note that the grammar is still nondeterministic but, in practice, *first* and *follow* sets could be used to choose between alternative transitions.

(PDAs to grammars) (See also Martin, Section 13.2) Let $P = (Q, \Sigma, \Lambda, \delta, q_0, Z_0)$ be a PDA. Define a CFG $G = (V, \Sigma, R, S)$ such that $L(G) = N(P)$ as follows. The set of variables V consists of the special symbol S, which is the start symbol, and all symbols of the form $[pXq]$, where $p, q \in Q$ and $X \in \Gamma$. The productions of G are the following:

1. For each state p,

$$S \to [q_0 Z_0 p].$$

2. For each state p, symbol $a \in \Sigma \cup \{\varepsilon\}$ and stack symbol A such that $\delta(p, a, A)$ contains (q, ε),

   $$[pAq] \to a.$$

3. For each state p, symbol $a \in \Sigma \cup \{\varepsilon\}$ and stack symbol A, if $\delta(p, a, A)$ contains $(q, Y_1 \cdots Y_k)$, where $k \geq 1$, then for all $r_1, \ldots, r_k \in Q$,

   $$[pXr_k] \to a[rY_1r_1][r_1Y_2r_2] \cdots [r_{k-1}Y_kr_k].$$

The idea is that $[pXq]$ derives all strings which from state p with A on top of the stack, lead (eventually) to state q with A having (perhaps indirectly) been removed from the stack.

More formally, we can prove that

$$[pXq] \overset{*}{\Rightarrow} w \text{ if and only if } (p, w, X) \vdash^* (q, \varepsilon, \varepsilon).$$

In practice, this construction yields a grammar that can be greatly simplified; see the start of the next section.

See H, Example 6.15, which is really too simple to illustrate the method.

Example. Consider the language $\{0^n1^n \mid n \geq 1\}$ again. It can be recognised by a PDA $P = (\{p, q\}, \{0, 1\}, \{X, Z\}, \delta, p, Z)$, where

$$\begin{aligned}
&\delta(p, 0, Z) = \{(p, XZ)\}\\
&\delta(p, 0, X) = \{(p, XX)\}\\
&\delta(p, 1, X) = \{(q, \varepsilon)\}\\
&\delta(q, 1, X) = \{(q, \varepsilon)\}\\
&\delta(q, \varepsilon, Z) = \{(q, \varepsilon)\}
\end{aligned}$$

The corresponding grammar is:

$$\begin{aligned}
&S \to [pZp]\\
&S \to [pZq]\\
&[pXq] \to 1\\
&[qXq] \to 1\\
&[qZq] \to \varepsilon\\
&[pZp] \to 0[pXp][pZp]\\
&[pZp] \to 0[pXq][qZp]\\
&[pZq] \to 0[pXp][pZq]\\
&[pZq] \to 0[pXq][qZq]\\
&[pXp] \to 0[pXp][pXp]\\
&[pXp] \to 0[pXq][qXp]\\
&[pXq] \to 0[pXp][pXq]\\
&[pXq] \to 0[pXp][pXq]
\end{aligned}$$

Definition of a deterministic PDA (DPDA): Informally, no choice from any ID. Formally:

1. For all $q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $X \in \Gamma$, $\delta(q, a, X)$ has at most one element.
2. For all $q \in Q$, $X \in \Gamma$, if $\delta(q, a, X)$ is nonempty for some $a \in \Sigma$, then $\delta(q, \varepsilon, X)$ is empty.

Example. L_{wwr} does *not* have a DPDA (that accepts by final state), but $L_{wcwr} = \{ wcw^R \mid w \in \{0, 1\}^* \}$ *does* have a DPDA (Ex. 6.16). Note that L_{wcwr} is not regular.

Example. $\{ w \in \{a, b\}^* \mid w = w^R \}$ also does not have a DPDA (that accepts by final state).

Exercise. Construct a DPDA for the previous LL(1) grammar for expressions.

Note that strictly fewer languages are recognised by DPDAs that accept by empty stack than are recognised by DPDAs that accept by final state.

If L is a regular language, then $L = L(P)$ for some DPDA P (H, Theorem 6.17). Basically, let P be the DFA for L, regarded as a DPDA that ignores its stack.

Note that it is *not* the case that if L is regular then $L = N(P)$ for some DPDA P. I.e., not every regular language is the language of a DPDA that accepts by empty stack.

Example. $\{0\}^*$ is not $N(P)$ for any DPDA P.

Similarly, not every CFL L satisfies $L = L(P)$ for some DPDA P that accepts by final state. (This is not so easy to prove; see H, Exercise 6.4.4.)

In summary, the following strict inclusions hold:

> regular languages $\subset$ languages accepted by DPDAs (by final state) $\subset$ CFLs

If a language L is recognised by a DPDA that accepts either by empty stack or by final state, then L has an unambiguous CFG (H, Theorems 6.20 and 6.21). I.e.:

> regular languages $\subset$ languages accepted by DPDAs $\subset$ languages with unambiguous CFGs $\subset$ CFLs

Note that each of the inclusions in the above chain is strict.

Exerecise. Give examples to demonstrate that each inclusion is strict.

The class of languages that have a DPDA (that accepts by final state) is called the class of LR(k) languages (Knuth). Every LR(k) grammar is unambiguous. The parser generator Yacc can parse the LALR(1) subclass of these languages. We then have the more refined inclusions:

> regular languages $\subset$ LL(1) languages $\subset$ LR(k) languages
> regular languages $\subset$ SLR(1) languages $\subset$ LALR(1) languages $\subset$ LR(k) languages
> LR(k) languages = DPDA languages $\subset$ CFLs

(The LL(1) and LALR(1) languages are incomparable.)

Deterministic PDAs do not have unique minimal DPDAs.

Exercise. Construct a DPDA with two nonidentical, minimal DPDAs.

Properties of context-free languages (H, Chapter 7; S, Section 3.3)

It is often possible to *simplify* CFGs.

Theorem. If G is a CFG generating a language not equal to $\emptyset$ or $\{\varepsilon\}$, then there is another CFG G' such that $L(G') = L(G) - \{\varepsilon\}$ and G' has no ε-productions ($A \rightarrow \varepsilon$), no unit productions ($A \rightarrow B$), and no useless symbols (every symbol occurs in some derivation of the form $S \overset{*}{\Rightarrow} \alpha A \beta \overset{*}{\Rightarrow} w \in \Sigma^*$).

A grammar G is in *Chomsky Normal Form* (CNF) if every production in G has the form $A \rightarrow a$ or $AtoBC$, where a is in Σ and B and C are in V, and G has no useless symbols.

Theorem. If G is a CFG generating a language not equal to $\emptyset$ or $\{\varepsilon\}$, then there is a grammar G' in Chomsky Normal Form such that $L(G') = L(G) - \{\varepsilon\}$.

See Sipser, pp.107–109, for the transformation to CNF.

A language is in *Greibach Normal Form* (GNF) if every production in G has the form $A \rightarrow aX_1 \dots X_n$ for $n \geq 0$, where $a \in \Sigma$ and $X_1, \dots, X_n \in V$.

Theorem. If G is a CFG generating a language not equal to $\emptyset$ or $\{\varepsilon\}$, then there is a grammar G' in Greibach Normal Form such that $L(G') = L(G) - \{\varepsilon\}$.

The transformation to GNF is not required for the course.

Proving languages are not context-free

Pumping lemma for CFLs (H, 7.2.2, the $uvwxy$ theorem; S, Theorem 2.34): Let L be a CFL. Then there exists a constant $n \geq 1$ such that, for every string z in L such that $|z| \geq n$, we can write $z = uvwxy$ in such a way that:

1. $|vwx| \leq n$ (the middle section is not too long).
2. $vx \neq \varepsilon$ (at least one of the strings to pump is not empty).
3. For all $k \geq 0$, the string uv^iwx^iy is in L (the strings v and x may be pumped any number of times, including 0, and the resulting string is still in L).

Proof outline. See H, Figs. 7.5 and 7.6. Assume the grammar for L is in CNF. Then every long string must have a long path in its parse tree. This path must contain some repeated variable. Fig. 7.7 then indicates how this allows substrings v and x to be pumped arbitrarily often.

Applications. The following languages are not context-free:

1. (H, Ex. 7.19) $L = \{\, a^kb^kc^k \mid k \geq 1 \,\}$. Suppose L were context-free. Then there exists n by the pumping lemma. Choose $z = a^nb^nc^n$. Let $z = uvwxy$. By the pumping lemma, $|vwx| \leq n$ and $vx \neq \varepsilon$. Then vwx cannot contain both as and cs. Suppose vwx contains only as and bs. Then vx contains only as and bs and has at least one of these symbols. By the pumping lemma, uwy is also in L ($i = 0$). But this string has fewer than n as or fewer than n bs (but still n cs), so it is not in L. Alternatively, suppose vwx contains only bs and cs. Then a similar argument shows that uwy both belongs to L and does not belong to L. In either case we have a contradiction, which shows that L cannot be context-free.

2. (H, Ex. 7.20) $L = \{\, 0^i1^j2^i3^j \mid i, j \geq 1 \,\}$. This time, choose $z = 0^n1^n2^n3^n$, and apply a similar argument.

3. (H. Exercise 7.2.1) $L = \{\, a^ib^jc^k \mid i < j < k \,\}$. Let n be the pumping-lemma constant and consider string $z = a^nb^{n+1}c^{n+2}$. We may write $z = uvwxy$, where v and x, may be

"pumped," and $|vwx| \leq n$. If vwx does not have cs, then uv^3wx^3y has at least $n+2$ as or bs, and thus could not be in the language.

If vwx has a c, then it could not have an a, because its length is at most n. Thus, uwy has n as, but no more than $2n+2$ bs and cs in total. Thus, it is not possible that uwy has more bs than as and also has more cs than bs. We conclude that uwy is not in the language, and now have a contradiction no matter how z is broken into $uvwxy$.

Closure properties of CFLS

A *substitution* s is a mapping from some alphabet Σ to a set of languages S, i.e., for each a in Σ, $s(a)$ is a language in S. Substitutions can be extended from symbols to strings and then to languages in the natural way.

If L is a CFL over alphabet Σ, and s is a substitution on Σ such that $s(a)$ is a CFL for each a in Σ, then $s(L)$ is a CFL (H, Theorem 7.23).

Theorem. The class of CFLs is closed under the following operations: union, concatenation, closure (*) and positive closure (+), and reversal. There are simple proofs based on operations on CFGs and, for the first three, on the above property of substitutions.

The class of CFLs is *not* closed under intersection.

See H, Example 7.26 ($L = \{\, 0^n1^n2^n \mid n \geq 0 \,\}$) for a counterexample.

It follows that the class of CFLs is also not closed under complementation or difference (H, Theorem 7.29).

However, the intersection of a CFL and a regular language *is* a CFL (H, Theorem 7.27). The proof involves the construction of a PDA for the intersection by a product construction, similar to that used to prove the regular languages are closed under intersection.

It follows that if L is a CFL and R is a regular language, then $L - R$ (= $L \cap R'$) is a CFL.

These results can also be used to show that given languages are or are not context-free.

Decision problems for CFLs (7.4)

The following properties of CFLs are decidable:

1. (Emptiness) Is the CFL L empty? (See 7.1.2.) Let $G = (V, \Sigma, R, S)$ be a grammar, and perform the following induction. Every symbols of Σ is "generating" (nonempty). If there exists a production $A \rightarrow \alpha$ in R and every symbol in α is generating then A is generating. If S is generating then $L(G)$ is nonempty. This algorithm is $O(n^2)$ in the length of G.

   Section 7.4.3 presents a more efficient algorithm that is $O(n)$ in the length of G.

2. (Finiteness) Is the CFL L finite?

3. (Membership) Does string w belong to the CFL L? The CYK algorithm solves this problem, and is $O(n^{3)}$ in the length of w.

   The CYK algorithm assumes a grammar $G = (V, \Sigma, R, S)$ for L in CNF. It constructs a triangular table from $w = a_1a_2 \ldots a_n$ as shown in Fig. 7.12. In this table, entry X_{ij} is the set of variables A such that $A \overset{*}{\Rightarrow} a_ia_{i+1} \ldots a_j$. The table is constructed from the bottom row upwards. If S is in X_{1n}, then $S \overset{*}{\Rightarrow} w$, so $w \in L$.

   If L has an LL(1) grammar, or more generally an LR(k) grammar, then there exists a linear-time algorithm to determine membership.

On the other hand, the following properties of CFLs are undecidable:

1. Is a given CFG unambiguous?
2. Is a given CFL inherently unambiguous?
3. Is the intersection of two given CFLs empty?
4. Are the languages of two given CFGs (or PDAs) equal?
5. Is the language of a given CFG equal to Σ^*?

Note, however, that the eqivalence of two given DPDAs *is* decidable.