Finite automata theory and formal languages (DIT321, TMV027)

Nils Anders Danielsson

2019-03-04

Today

- Closure properties for context-free languages.
- Some algorithms for context-free languages.
- Some undecidable problems.

Closure properties

Context-free languages

A language L over the alphabet Σ is *context-free* if there exists a context-free grammar G, with alphabet Σ, for which $L(G) = L$.

Context-free languages

- Every regular language is context-free.
- Exercise: Prove this.

Substitutions

Assume that

- Σ_1 and Σ_2 are alphabets and
- $F \in \Sigma_1 \rightarrow \wp(\Sigma_2^*)$.

The function F maps symbols to languages.
It can be lifted to strings and languages:

$$\begin{aligned} F &\in \Sigma_1^* \rightarrow \wp(\Sigma_2^*) \\ F(\varepsilon) &= \{\, \varepsilon \,\} \\ F(aw) &= F(a)F(w) \end{aligned}$$

$$\begin{aligned} F &\in \wp(\Sigma_1^*) \rightarrow \wp(\Sigma_2^*) \\ F(L) &= \bigcup_{w \in L} F(w) \end{aligned}$$

What is $F(\{\,01\,\}^*)$ when $F(0) = \{\,a\,\}$ and $F(1) = \{\,b, c\,\}$?

1. $\{\,a, b, c\,\}^*$
2. $\{\,abc\,\}^*$
3. $\{\,ab, ac\,\}^*$
4. $\{\,ac, bc\,\}^*$
5. $\{\,a\,\}^*\{\,b, c\,\}^*$
6. $\{\,a, b\,\}^*\{\,c\,\}^*$
7. $\{\,a\,\}^*\{\,bc\,\}^*$
8. $\{\,ab\,\}^*\{\,c\,\}^*$

Closure under substitutions

If

- Σ_1 and Σ_2 are alphabets,
- $L \subseteq \Sigma_1^*$ is context-free,
- $F \in \Sigma_1 \to \wp(\Sigma_2^*)$, and
- $F(a)$ is context-free for every $a \in \Sigma_1$,

then $F(L)$ is context-free.

Closure under substitutions

Idea:

- Replace each terminal a in a grammar for L with the start symbol of a grammar for $F(a)$.

Closure under union

- If L_1 and L_2 are context-free, then $L_1 \cup L_2$ is context-free.
- Substitute L_i for i in $\{ 1, 2 \}$.

Closure under concatenation

- If L_1 and L_2 are context-free, then $L_1 L_2$ is context-free.
- Substitute L_i for i in $\{\, 12 \,\}$.

Closure under Kleene star

- If L is context-free,
  then L^* is context-free.
- Substitute L for 1 in $\{ 1 \}^*$.

Closure under Kleene plus

- If L is context-free,
  then L^+ is context-free.
- Substitute L for 1 in $\{ 1 \}^+$.

Homomorphisms

Assume that

- Σ_1 and Σ_2 are alphabets and
- $h \in \Sigma_1 \to \Sigma_2^*$.

The function h maps symbols to strings.
It can be lifted to strings and languages:

$$\begin{aligned} h &\in \Sigma_1^* \to \Sigma_2^* \\ h(\varepsilon) &= \varepsilon \\ h(aw) &= h(a)h(w) \end{aligned}$$

$$\begin{aligned} h &\in \wp(\Sigma_1^*) \to \wp(\Sigma_2^*) \\ h(L) &= \{\, h(w) \mid w \in L \,\} \end{aligned}$$

The function $h \in \Sigma_1^* \to \Sigma_2^*$ is a
string homomorphism.

Closure under homomorphism

- If $L \subseteq \Sigma_1^*$ is context-free,
  then $h(L)$ is context-free.
- Apply the substitution $F(a) = \{ h(a) \}$ to L.

Prove that $\{ \, 01^n 23^n 45^n 6 \mid n \in \mathbb{N} \, \}$ is not a context-free language over $\{ \, 0, 1, 2, 3, 4, 5, 6 \, \}$.

You may use the fact that $\{ \, 0^n 1^n 2^n \mid n \in \mathbb{N} \, \}$ is not a context-free language over $\{ \, 0, 1, 2 \, \}$.

Closure under intersection

- If L_1 and L_2 are context-free,
  then $L_1 \cap L_2$ is *not* necessarily context-free.
- If L_1 and L_2 are context-free,
  then $L_1 \setminus L_2$ is *not* necessarily context-free.
- If L is a context-free language over Σ,
  then $\overline{L} = \Sigma^* \setminus L$ is *not* necessarily context-free.

Closure under intersection

- If L is context-free and R is regular, then $L \cap R$ is context-free.
- If L is context-free and R is regular, then $L \setminus R$ is context-free.

If Σ is an alphabet, $R \subseteq \Sigma^*$ is regular and $L \subseteq \Sigma^*$ is context-free, what can we say about $R \setminus L$?

1. It is always regular.
2. It is not necessarily regular,
   but always context-free.
3. It is not necessarily context-free.

Some algorithms

Testing emptiness

For any context-free language L,
given as a context-free grammar $G = (N, \Sigma, P, S)$,
we can decide if $L = \emptyset$:

- A symbol $X \in N \cup \Sigma$ is *generating* if $X \Rightarrow^* w$ for some $w \in \Sigma^*$.
- $L = \emptyset$ if and only if S is not generating.

Computing the generating symbols

The set of generating symbols can be computed (perhaps inefficiently) in the following way:

- Let the function $step \in \wp(N \cup \Sigma) \to \wp(N \cup \Sigma)$ be defined by

$$step(\Gamma) = \left\{ A \;\middle|\; \begin{array}{l} A \to \alpha \in P, \\ \text{every symbol in } \alpha \text{ is in } \Gamma \end{array} \right\}.$$

- Initialise Γ to Σ.
- Repeat until $step(\Gamma) \subseteq \Gamma$:
  - Set Γ to $\Gamma \cup step(\Gamma)$.
- Return Γ.

Testing if the empty string is a member

For any context-free language L,
given as a context-free grammar $G = (N, \Sigma, P, S)$,
we can decide if $\varepsilon \in L$:

- A nonterminal $A \in N$ is *nullable* if $A \Rightarrow^* \varepsilon$.
- We have $\varepsilon \in L$ if and only if S is nullable.

Computing the nullable nonterminals

The set of nullable nonterminals can be computed (perhaps inefficiently) in the following way:

- Let the function $step \in \wp(N) \rightarrow \wp(N)$ be defined by

$$step(E) = \left\{ A \;\middle|\; \begin{array}{l} A \rightarrow \alpha \in P, \\ \text{every symbol in } \alpha \text{ is a} \\ \text{nonterminal in } E \end{array} \right\}.$$

- Initialise E to $\emptyset$.
- Repeat until $step(E) \subseteq E$:
  - Set E to $E \cup step(E)$.
- Return E.

Let (N, Σ, P, S) be a context-free grammar in Chomsky normal form and a a terminal in Σ.

Fill in the missing pieces so that the following algorithm computes the set $\{\, A \in N \mid w \in \Sigma^*, A \Rightarrow^* aw \,\}$.

- Let the function $step \in \wp(N) \to \wp(N)$ be defined by $step(F) =$???.
- Initialise F to ???.
- Repeat until $step(F) \subseteq F$:
  - Set F to $F \cup step(F)$.
- Return F.

The CYK algorithm

For any context-free language L,
given as a context-free grammar G,
and for any nonempty string $w \in \Sigma^*$,
we can decide if $w \in L$.

The CYK algorithm

- Convert G to a grammar $G' = (N, \Sigma, P, S)$ in Chomsky normal form.
- Build a CYK table T for G' and w:
  - Let w_i denote the i-th symbol in w (counting from 1).
  - $T_{i,j}$ is defined for $i, j \in \{1, ..., |w|\}$ satisfying $i \leq j$.
  - $T_{i,j} = \left\{ A \in N \;\middle|\; A \Rightarrow^* w_i ... w_j \right\}$.
- Check if $S \in T_{1,|w|}$.

The CYK algorithm

The table can be computed in the following way:

- First set

$$T_{i,i} = \{\ A \mid A \to w_i \in P\ \}$$

  for each $i \in \{1, ..., |w|\}$.

- Then set

$$T_{i,j} = \left\{\ A \ \middle|\ \begin{array}{l} k \in \{\ i, ..., j-1\ \}, \\ B \in T_{i,k}, C \in T_{k+1,j}, \\ A \to BC \in P \end{array} \right\}$$

  for all $i, j \in \{1, ..., |w|\}$ satisfying $j - i + 1 = 2$.

- Repeat the previous step for $j - i + 1 = 3$, 4 and so on up to $|w|$.

The CYK algorithm

An example of dynamic programming.

Consider the following CYK table:

$\{S\}$			
$\emptyset$	$\{T\}$		
$\emptyset$	$\{S\}$	$\emptyset$	
$\{T, Z\}$	$\{U, O\}$	$\{U, O\}$	$\{T, Z\}$
0	1	1	0

Construct a parse tree for the string 0110, given the information that at least the following productions exist in the grammar:
$S \to ZT$, $S \to OU$, $T \to SZ$.

The CYK algorithm

- A potential problem:
  The size of G' can be quadratic
  in the size of G.
- A variant of the algorithm that does not use the $\textsc{Unit}$ transformation can be devised:
  - Time complexity: $O(|G||w|^3)$.
  - Space complexity: $O(|G||w|^2)$.

  See Lange and Leiß.

Some undecidable problems

Some undecidable problems

The following things cannot, in general, be determined (using, say, a Turing machine):

- If a context-free grammar is ambiguous.
- If a context-free language, given by a context-free grammar, is *inherently* ambiguous.
- If $L(G_1) = L(G_2)$ for two context-free grammars G_1 and G_2.
- ...

Some undecidable problems

If you want to know more about why certain problems are undecidable, then you might be interested in the course *Computability* (formerly known as “Models of computation”).

Today

- Closure properties for context-free languages.
- Some algorithms for context-free languages.
- Some undecidable problems.

Next lecture

- Pushdown automata.
- Turing machines.

- Deadline for the next quiz: 2019-03-07, 10:00.
- Deadline for the sixth assignment: 2019-03-10, 23:59.