

Lecture 9 – Closure properties of context-free languages, Dyck languages

NTIN071 Automata and Grammars

Jakub Bulín (KTIML MFF UK)

Spring 2025

** Adapted from the Czech-lecture slides by Marta Vomlelová with gratitude.
The translation, some modifications, and all errors are mine.*

Recap of Lecture 8

- Pushdown automata accept exactly context-free languages (constructions: CFG to PDA and PDA to CFG)
- A deterministic pushdown automaton (DPDA)
- DPDA recognize a proper subclass of context-free languages, accepts by empty stack iff prefix-free and accepts by final state (Deterministic PDA + acceptance by empty stack does not even cover regular languages!)
- Deterministic PDA have unambiguous grammars
- The landscape of languages
- Converting between representations of context-free languages
- Undecidable problems about context-free languages (preview)

2.12 Closure properties of context-free languages

Closed under union, concatenation, iteration, reverse

Theorem

If $L, L' \subseteq \Sigma^$ are context-free, then so are $L \cup L'$, $L.L'$, L^* , L^+ , L^R .*

Proof: Let G, G' be CFG generating L, L' such that $V \cap V' = \emptyset$.
Take a new start symbol $S_{new} \notin V \cup V'$.

- **union** $L \cup L'$: add the rule $S_{new} \rightarrow S_1 \mid S_2$
- **concatenation** $L.L'$: add $S_{new} \rightarrow S_1 S_2$
- **iteration** L^* : add $S_{new} \rightarrow SS_{new} \mid \epsilon$
- **positive iteration** L^+ : add $S_{new} \rightarrow SS_{new} \mid S$
- **reverse** L^R : reverse the bodies of all production rules
(i.e., $A \rightarrow \beta$ becomes $A \rightarrow \beta^R$)

□

Substitution and homomorphism

Recall the definitions

A (string) **substitution** is a mapping $\sigma: \Sigma^* \rightarrow \mathcal{P}(Y^*)$ where

- Σ and Y are finite alphabets, $Y = \bigcup_{x \in \Sigma} Y_x$
- for each $x \in \Sigma$, $\sigma(x)$ is a language over Y_x
- $\sigma(\epsilon) = \{\epsilon\}$ and $\sigma(u.v) = \sigma(u).\sigma(v)$

For a language $L \subseteq \Sigma^*$, $\sigma(L) = \bigcup_{w \in L} \sigma(w) \subseteq Y^*$.

A (string) **homomorphism** is defined similarly but each letter is mapped to a single word, $h: \Sigma^* \rightarrow Y^*$ where $h(x) \in Y_x^*$ for $x \in \Sigma$, $h(\epsilon) = \epsilon$ and $h(u.v) = h(u).h(v)$. Then $h(L) = \{h(w) \mid w \in L\}$.

The **inverse homomorphism** applied to a language $L' \subseteq Y^*$:

$$h^{-1}(L') = \{w \in \Sigma^* \mid h(w) \in L'\}$$

Example: substitution

Example

Consider $G = (\{E\}, \{a, +, (,)\}, \{E \rightarrow E + E \mid (E) \mid a\}, E)$. Let us have the following substitution:

- $\sigma(a) = L(G_a)$, where

$$G_a = (\{I\}, \{a, b, 0, 1\}, \{I \rightarrow I0 \mid I1 \mid Ia \mid Ib \mid a \mid b\}, I)$$

- $\sigma(+) = \{-, *, :, \div, \text{ mod } \}$
- $\sigma(() = \{ \{ \}$
- $\sigma() = \{ \}$

Take $(a + a) + a \in L(G)$. Note that $(a + a) + a \notin \sigma(L(G))$, because $+ \notin \sigma(+)$. But e.g. $(a001 - bba) * b1 \in \sigma((a + a) + a) \subseteq \sigma(L(G))$

What if we modify the definition: $\sigma(() = \{ (, [, \sigma() = \{),] \}$?

Example: homomorphism

Example

$G = (\{E\}, \{a, +, (,)\},$
 $\{E \rightarrow E + E \mid (E) \mid a\}, E)$

- $h(a) = \epsilon$
- $h(+) = \epsilon$
- $h(() = \text{left}$
- $h() = \text{right}$

- $h((a + a) + a) = \text{leftright},$
- $h^{-1}(\text{leftright}) \ni (a + +)a.$

Example

$G = (\{E\}, \{a, +, (,)\},$
 $\{E \rightarrow E + E \mid (E) \mid a\}, E)$

- $h_2(a) = a$
- $h_2(+) = +$
- $h_2(() = \epsilon$
- $h_2() = \epsilon$

Are the following regular?

- $L(G)$
- $h_2(L(G))$
- $h_2^{-1}(h_2(L(G)))$

Is $h_2^{-1}(h_2(L(G))) = L(G)$?

Closure under substitution and homomorphism

Theorem

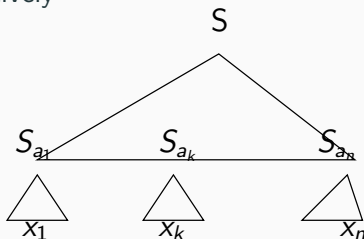
Let $L \subseteq \Sigma^*$ be a context-free language.

- (i) If σ is a substitution on Σ such that $\sigma(a)$ is context-free for all $a \in \Sigma$, then $\sigma(L)$ is context-free.
- (ii) If h is a homomorphism on Σ , then $h(L)$ is context-free.

Idea: replace terminals $a \in T$ in a parse tree with corresponding parse trees for $\sigma(a)$ or $h(a)$, respectively

Proof: (ii) follows immediately from (i), define $\sigma(a) = \{h(a)\}$

(i) construct a context-free grammar for $\sigma(L)$ [next slide]



Proof: construct the grammar for $\sigma(L)$

Let us have the following context-free grammars, assume all their variable sets are (pairwise) disjoint:

- $G = (V, T, \mathcal{P}, S)$ generating L
- $G_a = (V_a, T_a, \mathcal{P}_a, S_a)$ generating $\sigma(a)$, for $a \in T$

Construct a grammar $G' = (V', T', \mathcal{P}', S')$ where

- $V' = V \cup (\bigcup_{a \in T} V_a)$
- $T' = \bigcup_{a \in \Sigma} T_a$
- $\mathcal{P}' = (\bigcup_{a \in \Sigma} \mathcal{P}_a) \cup \mathcal{P}''$ where \mathcal{P}'' is obtained from \mathcal{P} by replacing every terminal a by the variable S_a
- $S' = S$

Clearly, G' generates the language $\sigma(L)$.

□

Example

Consider the language

$$L = \{a^i b^j \mid 0 \leq i \leq j\}$$

and the following substitution:

- $\sigma(a) = L_a = \{c^i d^i \mid i \geq 0\}$
- $\sigma(b) = L_b = \{c^i \mid i \geq 0\}$

The grammars G , G_a , and G_b :

- $S \rightarrow aSb \mid Sb \mid \epsilon$
- $S_a \rightarrow cS_a d \mid \epsilon$
- $S_b \rightarrow cS_b \mid \epsilon$

Then the grammar G' for $\sigma(L)$ consists of the following rules:

$$S \rightarrow S_a S S_b \mid S S_b \mid \epsilon, \quad S_a \rightarrow c S_a d \mid \epsilon, \quad S_b \rightarrow c S_b \mid \epsilon$$

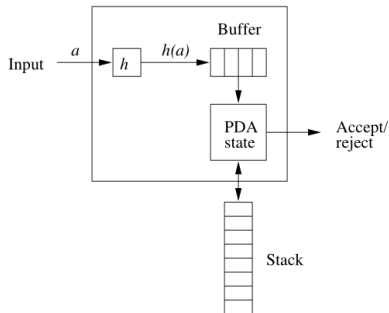
Closure under inverse homomorphism

Theorem

Let $L \subseteq Y^*$ be a context-free language and $h: \Sigma^* \rightarrow Y^*$ a homomorphism. Then $h^{-1}(L)$ is also context-free. Moreover, if L is deterministic, then so is $h^{-1}(L)$.

Idea: simulate a PDA M for L

- read a letter $a \Rightarrow$ place $h(a)$ into an inner buffer
- simulate M , but with input taken from the buffer
- if buffer empties, read the next letter from real input
- accept iff empty buffer and M is in accepting state



NB: buffer is **finite** \Rightarrow can be encoded in the states:

state = (state, buffer contents) 10

Proof: the construction

Let $M = (Q, Y, \Gamma, \delta, q_0, Z_0, F)$ (by final state). We define a PDA

$$M' = (Q', \Sigma, \Gamma, \delta', [q_0, \epsilon], Z_0, F \times \{\epsilon\})$$

where the set of states is the following (u is the buffer)

$$Q' = \{[q, u] \mid q \in Q, u \in Y^*, (\exists a \in \Sigma)(\exists v \in Y^*) h(a) = vu\}$$

and the transition function is defined as follows:

- **[re]fill buffer:**

$$\delta'([q, \epsilon], a, Z) = \{([q, h(a)], Z)\}$$

- **read from buffer:**

$$\begin{aligned} \delta'([q, u], \epsilon, Z) = & \{([p, u], \gamma) \mid (p, \gamma) \in \delta(q, \epsilon, Z)\} \\ & \cup \{([p, v], \gamma) \mid (p, \gamma) \in \delta(q, b, Z), u = bv\} \end{aligned}$$

For a DPDA M , the resulting M' is also deterministic. □

Closure properties: it's complicated

CFLs not closed under intersection

Example

$$L = \{0^n 1^n 2^n \mid n \geq 1\} = \{0^n 1^n 2^i \mid n, i \geq 1\} \cap \{0^i 1^n 2^n \mid n, i \geq 1\}$$

L is not context-free, even though both operands of the intersection are context-free:

$L_1 = \{0^n 1^n 2^i \mid n, i \geq 1\}$ generated by $G = (\{S, A, B\}, \{0, 1\}, \mathcal{P}, S)$ with production rules

$$\mathcal{P} = \{S \rightarrow AB, A \rightarrow 0A1 \mid \epsilon, B \rightarrow 2B \mid \epsilon\}$$

$L_2 = \{0^n 1^n 2^i \mid n, i \geq 1\}$ generated similarly using production rules

$$\mathcal{P} = \{S \rightarrow AB, A \rightarrow 0A \mid \epsilon, B \rightarrow 1B2 \mid \epsilon\}$$

Simulating two PDAs in parallel

Regular languages are closed under intersection, because we can simulate two DFAs in parallel. Why not PDAs?

- the FA units can be merged (same as for DFAs)
- reading input can be merged (one automaton can wait)
- but two stacks cannot be simulated on one stack!

In fact, 'PDAs with two stacks' are equivalent to **Turing machines**, can recognize any **recursively enumerable** language $L \in \mathcal{L}_0$.

⋮

But what if one of the PDAs does not really use its stack?

Intersection of a context-free and a regular language

Theorem

Let L be a context-free language and R a regular language. Then $L \cap R$ is context free. Moreover, if L is deterministic, so is $L \cap R$.

Proof: Let $L = L(P)$ for a PDA $P = (Q_1, \Sigma, \Gamma, \delta_1, q_1, Z_0, F_1)$ and $R = L(A)$ for a DFA $A = (Q_2, \Sigma, \delta_2, q_2, F_2)$. Construct a PDA

$$M = (Q_1 \times Q_2, \Sigma, \Gamma, \delta, (q_1, q_2), Z_0, F_1 \times F_2)$$

where we have a transition $\delta((q_1, q_2), a, X) \ni ((r_1, r_2), \gamma)$ iff either

- (i) $a \neq \epsilon$ and $(r_1, \gamma) \in \delta_1(q_1, a, X)$ and $r_2 = \delta_2(q_2, a)$, or
- (ii) $a = \epsilon$ and $(r_1, \gamma) \in \delta_1(q_1, \epsilon, X)$ and $r_2 = q_2$

In (i) both automata read input, in (ii) P works on its stack while A waits. Clearly, $L(M) = L(P) \cap L(A)$ (P and R run in parallel). \square

An application: proving non-context-freeness

Example

$L = \{0^i 1^j 2^k 3^\ell \mid i = 0 \text{ or } j = k = \ell\}$ is not context-free.

By contradiction, assume L is context-free.

The language $L_1 = \{01^j 2^k 3^\ell \mid i, j, k \geq 0\}$ is regular (e.g. a regular grammar $\{S \rightarrow 0B, B \rightarrow 1B \mid C, C \rightarrow 2C \mid D, D \rightarrow 3D \mid \epsilon\}$).

But $L \cap L_1 = \{01^i 2^i 3^i \mid i \geq 0\}$ is not context-free, a contradiction with the previous theorem. \square

In fact, L is a **context-sensitive** language:

$S \rightarrow \epsilon \mid 0 \mid 0A \mid B_1 \mid C_1 \mid D_1$	$DC \rightarrow CD$ rewrite as	$1C \rightarrow 12$
$A \rightarrow 0 \mid 0A \mid P, P \rightarrow 1PCD \mid 1CD$	context-sensitive rules	$2C \rightarrow 22$
$B_1 \rightarrow 1 \mid 1B_1 \mid C_1$	$DC \rightarrow XC, XC \rightarrow XY,$	$2D \rightarrow 23$
$C_1 \rightarrow 2 \mid 2C_1 \mid D_1$	$XY \rightarrow CY, CY \rightarrow CD$	$3D \rightarrow 33$
$D_1 \rightarrow 3 \mid 3D_1$		

CFLs are not closed under difference nor complement

Theorem

The class of the context-free languages is not closed under difference, nor complement.

Proof: $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$, closure under complement would imply closure under intersection. For difference, use $\overline{L} = \Sigma^2 - L$. \square

NB: PDA is non-deterministic, switching accepting/non-accepting states does not work.

Proposition

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

Proof: $L - R = L \cap \overline{R}$, and \overline{R} is also regular. \square

DCFLs are closed under complement

Theorem

The complement of a deterministic CFL is also deterministic.

Proof: Idea: accept iff the original DPDA rejects.

But we need to:

- catch failure due to empty stack by a new bottom of the stack
- recognize possible ϵ -transition cycle by a counter
- at the end of input, check if we are in an accepting state, or transitioned out of it using ϵ -transitions; in that case, reject



DCFLs are not closed under union nor intersection

Recall: intersection of a DCFL and a regular language is a DCFL

Example

$L = \{a^i b^j c^k \mid i \neq j \text{ or } j \neq k \text{ or } i \neq k\}$, a union of three DCFLs, is context-free but not a DCFL.

Proof: $\bar{L} \cap L(a^* b^* c^*) = \{a^i b^j c^k \mid i = j = k\}$ would be a DCFL; it's not even context-free (Pumping lemma). \square

Note: This also implies that DCFLs are not closed under intersection (de Morgan laws: $L_1 \cup L_2 = \overline{\bar{L}_1 \cap \bar{L}_2}$).

DCFLs are not closed under homomorphism

Example

Consider languages $L_1 = \{a^i b^j c^k \mid i \neq j\}$, $L_2 = \{a^i b^j c^k \mid j \neq k\}$, $L_3 = \{a^i b^j c^k \mid i \neq k\}$ which are deterministic context-free.

- The language $0L_1 \cup 1L_2 \cup 2L_3$ is a DCFL, construct a DPDA.
- The language $L_1 \cup L_2 \cup L_3$ is not a DCFL, otherwise also $\overline{L_1 \cup L_2 \cup L_3} \cap L(a^* b^* c^*) = \{a^i b^j c^k \mid i = j = k\}$ would be.

But $h(0L_1 \cup 1L_2 \cup 2L_3) = L_1 \cup L_2 \cup L_3$ for the homomorphism:

- $h(0) = \epsilon$, $h(1) = \epsilon$, $h(2) = \epsilon$,
- $h(s) = s$ for all other symbols.

But recall: DCFLs are closed under inverse homomorphism.

Closure properties: summary

language	regular (RL)	context-free	deterministic CFL
union	YES	YES	NO
intersection	YES	NO	NO
\cap with RL	YES	YES	YES
complement	YES	NO	YES
homomorphism	YES	YES	NO
inverse hom.	YES	YES	YES

2.13 Dyck languages

Definition

The **Dyck language** D_n is defined over $\Sigma_n = \{a_1, a_1^|, \dots, a_n, a_n^|\}$ using the context-free grammar with the following rules:

$$S \rightarrow \epsilon \mid SS \mid a_1Sa_1^| \mid \dots \mid a_nSa_n^|$$

- the Dyck language D_n captures correctly parenthesized expressions with n types of parentheses
- we use it to describe computations of an arbitrary PDA
- to show that any context-free language can be expressed as:

$$L = h(D_n \cap R)$$

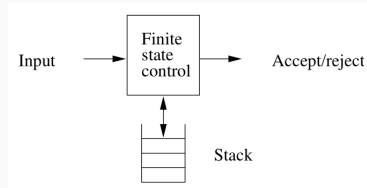
- the regular language R describes all computation steps
- the Dyck language selects only valid computations
- the homomorphism h cleans auxiliary symbols

A characterization of context-free languages

If we only push to the stack, **never pop**: it suffices to remember the top symbol; thus we only need **finite memory**, i.e. a FA

If we also need to pop:

- finite memory is not enough
- but memory access is not arbitrary, a stack is LIFO (last in–first out)



Expand the computation with a stack to a linear structure:

X symbol pushed

X^{-1} symbol popped

Pushing and popping form a pair that behaves like parentheses:

$$\underbrace{ZZ^{-1}} \underbrace{B \underbrace{AA^{-1}} \underbrace{CC^{-1}} B^{-1}}$$

The theorem

Theorem

For any context-free language L there exist a regular language R , a Dyck language D and a homomorphism h s.t. $L = h(D \cap R)$.

Proof: Start with a PDA recognizing L , accepting by empty stack. First, convert to instructions of the form:

$$\delta(q, a, Z) \in (p, w), |w| \leq 2$$

(Instructions pushing more symbols can be split using new states).

Let $R^|$ consist of all expressions of the form

$$q^{-1}aa^{-1}Z^{-1}BAp$$

for instruction $\delta(q, a, Z) \ni (p, AB)$, and similarly for instructions $\delta(q, a, Z) \in (p, A)$ and $\delta(q, a, Z) \in (p, \epsilon)$. (For $a = \epsilon$ omit aa^{-1} .)

Now define the regular language as $R = Z_0q_0(R^|)^*Q^{-1}$.

Proof cont'd

The Dyck language D is over the alphabet

$$\Sigma \cup \Sigma^{-1} \cup Q \cup Q^{-1} \cup \Gamma \cup \Gamma^{-1}$$

The language $D \cap R = D \cap Z_0 q_0 (R^I)^* Q^{-1}$ describes valid computations, e.g.

$$\underbrace{Z_0 \overbrace{q_0 q_0^{-1}} aa^{-1} Z_0^{-1}} B \underbrace{A \overbrace{pp^{-1}} bb^{-1} A^{-1} \overbrace{qq^{-1}} cc^{-1} B^{-1} \overbrace{rr^{-1}}}_{}$$

The homomorphism h selects the input word being read:

$$h(a) = a \quad \text{for input symbols } a \in \Sigma$$

$$h(y) = \epsilon \quad \text{for all other symbols}$$



Summary of Lecture 9

- Closure properties of context-free languages (including substitution, homomorphism, inverse homomorphism)
- Also closure properties of deterministic CFLs
- Dyck languages, a characterization of context-free languages