# FJA (IB005) - shrnuti

(Strucny a velmi neformalni vypis zakladnich algoritmu a definic, urceno k opakovani. Autor neruci za spravnost informaci.)

## REGULARNI JAZYKY

**Reg. gramatika:**
A → a nebo A → aB (plus S → eps., pokud S není nikde na prave strane)
Formalne ctverice G = ({neterminaly}, {abeceda}, pravidla, pocatecni neterminal)

**DFA - deterministic finite automaton:**
A = ({stavy}, {abeceda}, prechodova fce, pocatecni stav, {koncove stavy})

**Pumping lemma, pouziti pro dukaz neregularity:**
Necht n je libovolne. Zvolime slovo w z L, delka w >= n. Pro vsechna rozdeleni w = xyz, y ruzne od epsilon a delka xy <= n plati: Pro nejake i: pokud xy^iz nepatri do L, pak L neni regularni.
Napr. L = {a^nb^n, n > 0}
w = a^n.b^n
x = a^k, k >= 0
y = a^l, l > 0
z = a^(n-k-l).b^n
pro i = 2: xy^2z = a^k.a^l.a^l.a^(n-k-l).b^n = a^(n+l).b^n, coz nepatri do L, protoze l>=0
Z P. L. plyne, ze L neni regularni.

**Minimalizace:**
1. odstraneni nedosazitelnych stavu
2. ekvivalence stavu podle delky slova - v 1. kroku 1. skupina vsechny nekoncove stavy, 2. skupina vsechny koncove. Do nove tabulky zapiseme, do ktere skupiny se z ktereho stavu jde. Nasledne nova
tabulka a znovu udelame skupiny stavu podle toho, ktere vedou do stejnych skupin

**Kanonizace:**
1. Jako prvni dame pocatecni stav, pojmenovat A
2. Kazdy dalsi stav, na ktery pri prochazeni od pocatecniho narazime, pojmenujeme nasl. pismenem a pridame do tabulky.

**Prevod NFA na DFA:**
- Paralelni prochazeni po symbolech.
- pocatecni stav opiseme, ale z mnozin stavu, do kterych se lze dostat, udelame nove stavy.
- napr. kdyz ze stavu 1 se da dostat pod a do stavu 2,3, tak zalozime novy stav {2,3}. Z nej se pak pod a da dostat do sjednoceni stavu, do kterych se pod a dalo dostat ze stavu 2 a 3.

**Prevod eps-DFA na DFA:**
1. pro kazdy stav udelame epsilon okoli, D-e, tj. mnozinu stavu, kam se lze dostat pod epsilonem
2. pro kazdy symbol nejprve vytvorime mnozinu stavu X, kam se lze dostat z D-e daneho stavu pod aktualnim symbolem
3. Sjednotime D-e mnoziny stavu X
4. Oznacime nove koncove stavy - vsechny ty, ktere maji nejaky koncovy stav v D-e

**Prevod Regex na DFA:**

- q1 ->a+b q2 = q1 ->a,b q2 (pouze prepis)
- q1 ->a* q2 = q1 ->eps q3 ->a q3 ->eps q2 (vlozi se novy stav, do- a z- ktereho se jde pres eps. a na nem se cykli)
- q1 ->a.b q2 = q1 ->a q3 ->b q2 (rozlozi se pres novy stav na 2 prechody)

**Prevod DFA na Regex:**
1. a) pridame novy inicialni stav, z nej pod eps. do puv. inic. stavu
   b) udelame 1 koncovy stav q-f, do nej pod eps. ze vsech puvodnich. konc. stavu
2. postupne odstranujeme stavy a tvorime regex, nakonec 1 regex z inic. do koncoveho stavu

# BEZKONTEXTOVE JAZYKY (CFL)

**Uzaverove vlastnosti:**
vsechny zakladni krome: pruniku a doplnku

(pro DCFL: vsechny zakladni vcetne doplnku krome: sjednoceni a pruniku)

**CFG:**
A → X, delka X >= 1 (krome S → eps., pokud S není nikde na prave strane)

**CFG v CNF (Chomskeho normalni forma):**
A → BC nebo A → a
delka odvozeni 2n-1

**CFG v GNF (Greibachove normalni forma):**
A → a(BCD...)* (= neterminal + 0 nebo vice terminalu)

**Zasobnikovy automat PDA (push-down automaton):**
A = ({neterminaly}, {abeceda}, {zasobnikove symboly}, prechod. fce, inic. stav, inic. zas. symbol, {koncove stavy})

**Redukovana gramatika:**
1. odstranit nenormovane neterminaly, tj. takove, ktere nelze prevest na terminalni retezec
2. odstranit nedosazitelne neterminaly, tj. nelze na ne dojit z inic. neterminalu
Dulezite: zachovat poradi kroku!

**Odstraneni eps-pravidel:**
1. vytvorit mnozinu neterminalu N, ktere lze prepsat na epsilon (i ve vice krocich)
2. pridat vsechny kombinace pravidel vznikle odstranenim nektereho (nebo vice) prvku z N
(napr. S -> ABC, A-> a | aA | eps., B -> b | bB | eps., C -> c | cC | eps. pridame pro S: AB, AC, BC, A, B, C, eps. )
3. odstranit puvodni eps. pravidla

**Odstraneni jednoduchych pravidel (A -> B)**
1. pro kazdy neterminal X vytvorit mnozinu neterminalu N-x, na ktere se muze prepsat jedn. prav. (napr. S -> A | aB, N-s = {S, A})
2. pro kazdy neterminal X odstranit jednoducha pravidla a sjednotit vsechna ostatni pravidla pro vsechny neterminaly z N-x

**Odstraneni leve rekurze**

Prima:
X -> Xa | bY
---
X -> bY | bYX'
X' -> a | aX'
Neprima:
1. seradit neterminaly libovolne
2. pro kazdy neterminal udelej substituci vsech levostrannych vyskytu predchazejicich neterminalu
3. pro kazdy neterminal proved eliminaci prime leve rekurze

**Prevod do GNF:**
1. prevest na vlastni (redukovana, bez jednoduchych a bez eps. pravidel) a nelevorekurzivni
2. zkonstruovat usporadani, takove, ze je-li A -> BX, pak A < B
3. projit neterminaly odzadu a udelat substituci levostrannych neterminalu
4. vsechny terminaly na 2. a dalsi pozici zmenit na neterminaly (A -> aBc --> A -> aBc'; c' -> c)

**Prevod do CNF:**
1. odstranit eps. pravidla
2. odstranit jednoducha pravidla
3. vsechny pravidla jina nez A -> a ci A -> BC prepsat na pravidla typu A -> BC nasledovne:
A -> abCdE
---
A -> a'<bCdE>
a' -> a
<bCdE> -> b'<CdE>
b' -> b
<CdE> -> C<dE>
<dE> -> d'E
d' -> d

**Prevod CFG na PDA:**

**a) analyza shora dolu:**
PDA s jednim stavem. Neterminaly i terminaly (pripadne s carkou pro odliseni od abecedy) tvori zasobnikove symboly.
1. Nejprve nacte na zasobnik inic. neterminal
2. Kdykoliv muze neterminal na zasobniku prepsat pod eps. krokem dle pravidel gramatiky
3. Je-li na zasobniku terminal (resp. symbol znacici terminal), pak ho zkontroluje se slovem a pokud sedi, tak ho smaze a posune se ve slove.

**b) analyza zdola nahoru:**
Potrebujeme rozsireny PDA, ktery umi cist libovolny pocet symbolu na zasobniku. Obsahuje specialni symbol pro dno zasobniku, prechodova fce je definovana pro retezec zasob. symbolu.
1. Postupne nacita slovo a pridava symboly odpovidajicich terminalu
2. Kdykoliv je potreba (nedeterministicky), muze si pridat pocatecni neterminal
3. Snazi se prepisovat obsah zasobniku tak, aby se zkracoval a postupne na nem zustal jen pocatecni neterminal a dno. V takove chvili (je-li docteno slovo), tak prechazi do akceptujiciho stavu.

**Pumping lemma pro CFL:**
Priklad: L = {a^n.b^n.c^n}

Necht n je lib. Zvolime z = a^n.b^n.c^n, delka z >= n. Pro kazde u,v,w,x,y (z abecedy) splnujici z = u.v.w.x.y, delka vwx <= n, delka v.x > 0 plati:
1. vwx patri do {a,b}^+; nebo
2. vwx patri do {b,c}^+
Pro i = 0 plati u.v^i.w.x^i.y = u.w.y nepatri do L, protoze:
1. uwy = p.c^n, p patri do {a,b}^*, delka p = 2n - delka v.x, tedy zjevne p nemuze byt a^n.b^n
2. uwy = a^n.p, p patri do {b,c}^*, delka p = 2n - delka v.x, tedy zjevne p nemuze byt b^n.c^n
Z P. L. plyne, ze L neni CFL.

**Vlastnost sebevlozeni:**
CFG ma vlastnost sebevlozeni, pokud existuje neterminal: A =>* uAv (u,v patri do abecedy*)
CFL ma vlastnosti sebevlozeni, pokud kazda CFG, ktera ho generuje, ma vlastnost sebevlozeni.
Regularni jazyky nemaji vlastnost sebevlozeni.
Kazdy CFL, ktery neni regularni, ma vlastnost sebevlozeni.

## Kontextove jazyky CSL

CSL = context-sensitive language

**CSG**
X -> Y, |X| <= |Y|, jinak X a Y libovolne

**LBA = Linearly bounded automaton**
LBA M je desetice: (stavy, abeceda, paskove symboly, leva zarazka, prava zarazka, prazdne policko, prechodova fce, inic. stav, akcept. stav, zamitajici stav)
Omezeni oproti TM: muze pracovat jen na pasce delky vstupu
Pozn.: Neni znamo, jakou mnozinu jazyku akceptuje deterministicky LBA.

## Turingovy stroje, frazove gramatiky a Rek. + R.E. jazyky

**Uzavrenost:**
Rekurzivni: vse zakladni
R.E.: vse zakladni krome doplnku

**TM je devitice**: stavy, abeceda, paskove symboly, leva koncova znacka, prazdne policko, prechodova funkce, pocatecni stav, akceptujici stav, zamitajici stav

**Uplny TM**
- TM, ktery pro kazdy vstup zastavi, tzv. rozhoduje jazyk
- odpovida rekurzivnim jazykum, resp. rozhodnutelnym problemum

**(Neuplny) TM**
- akceptuje kazdy vstup z L(M), tedy pro kazdy vstup patrici do jazyka zastavi, tzv. akceptuje L
- pro vstupy, ktere nejsou z L, muze zastavit a zamitnout, ale take muze cyklit
- odpovida RE, resp. frazovym gramatikam, resp. castecne rozhodnutelnym problemum
(- castecne rozhodnutelne problemy jsou zaroven i nerozhodnutelne problemy)

**Postova veta**
Jazyk L je R.E. a co-L je R.E. <=> L je rekurzivni (a zrejme i co-L je rekurzivni).

**Redukce**
(A z abecedy F*, B z abecedy G*)
A se redukuje na B (A <= B) <=> Existuje fce f: F*->G*, ktera je rekurzivni a zachovava prislusnost, tzn. plati: w patri do A <=> f(w) patri do B.

Pouziti redukce:
A <= B:
B je rekurzivni (pripadne R.E.) => A je rekurzivni (R.E.)
A neni rekurzivni (pripadne R.E.) => B neni rekurzivni (pripadne R.E.)

PP = problem prislusnosti = {<M>#<w> | M akceptuje w}
PZ = HALT = problem zastaveni = {<M>#<w> | M zastavi na w}

**Dovetailing:**
Metoda "simultanniho" vypoctu na vice slovech. Zpusob, jak se vyhnout cykleni.
Pouziti napr. kdyz deterministicky hledame nejake slovo, ktere TM M akceptuje, a potrebujeme predejit cykleni na slovech predchazejicich.
Vypocet se provadi tak, ze postupne dela 1 krok nejprve v 1 slove, pak prida 2. slovo, udela 2. krok v 1. slove, 1. krok v 2. slove, prida 3. slovo a zase v kazdem slove jeden krok atd.

**Diagonalizace:**
Metoda k vyvraceni spocetnosti. Napr. pro cisla mezi 0 a 1: Predpokladejme, ze mame zapsana a ocislovana vsechna cisla z intervalu 0 a 1 pod sebou v nekonecne tabulce, tedy ze jde o spocetnou mnozinu. Nyni vytvorime nove cislo tak, ze bereme 1 cislo z tabulky za druhym a k 1. cislici 1. cisla pricteme 1 (mod 10), pak k 2. cislici 2. cisla 1 (mod 10), atd. tedy postupujeme po diagonale a vzdy pricteme 1 (mod 10), cimz zajistime, ze se nase nove cislo lisi od kazdeho cisla v tabulce nejmene v te jedne cislici. Takove nove cislo jde pridavat kdykoliv znova a znova, tedy zjevne dana mnozina neni spocetna.

**Postuv korespondencni problem, PKP a inPKP:**
Instance PKP: 2 seznamy neprazdnych slov nad stejnou abecedou: A = $x_1$, ..., $x_n$ a B = $y_1$, ..., $y_n$
Reseni dane instance je posloupnost cisel $i_1$, ..., $i_k$, takova, ze $x_{i1}$ $x_{i2}$ ... $x_{ik}$ = $y_{i1}$ $y_{i2}$ ... $y_{ik}$ .
Tedy vybereme z kazdeho seznamu postupne $i_1$-te slovo, $i_2$-te slovo, ... a chceme, aby nakonec vysly stejna slova.
PKP je formulovan jako trida problemu rozhodnout pro lib. instanci, zda ma reseni.

InPKP = inicialni PKP = PKP s modifikaci, ze jako prvni bereme vzdy 1. prvek.