BIK-AAG - Řešené příklady

Martin Plicka

October 24, 2012

1 Konečné automaty - názorně

Mějme následující automat ... zkuste si jej nakreslit.

	a	b	ϵ
$\rightarrow$ 0	{0,1}	{0,4}	{4}
1	{4,5}	{2}	{5}
2	{3}	{5,6}	{6}
$\leftarrow$ 3	{3}	{3}	{}
4	{}	{5}	{}
5	{6}	{}	{}
$\leftarrow$ 6	{6}	{6}	{}

Všimněte si, že hodnoty přechodové funkce jsou množiny - vzpomeňte si na definici nedeterministického automatu.

Postupně provedeme následující ...

1. Odstraníme epsilon přechody
2. Detererminizujeme
3. Minimalizujeme výsledný automat

1.1 Odstranění epsilon přechodu

V přednáškách lze algoritmus najít. Je potřeba se prokousat složitým výrazem. Nejprve je nutno zadefinovat $\varepsilon - CLOSURE$.

$$\varepsilon - CLOSURE(q) = \{p : (q, \varepsilon) \vdash^* (p, \varepsilon), p \in Q\}.$$

Česky řečeno, funkce pro každý stav vrací množinu stavů, do kterých se lze dostat pouze pomocí epsilon přechodů. Tato množina obsahuje alespoň výchozí stav q samotný. Je potřeba dát pozor a postupovat systematicky, stavy mohou být totiž dosažitelné sérií epsilon přechodů v řadě. Epsilonový uzávěr můžeme spočítat algoritmem podobným algoritmu na výpočet dosažitelných stavů, ovšem s modifikací na epsilon přechody.

V tomto příkladě je určení uzávěru jednoduché:
$\varepsilon - CLOSURE(0) = \{0, 4\}$
$\varepsilon - CLOSURE(1) = \{1, 5\}$
$\varepsilon - CLOSURE(2) = \{2, 6\}$
$\varepsilon - CLOSURE(3) = \{3\}$
$\varepsilon - CLOSURE(4) = \{4\}$
$\varepsilon - CLOSURE(5) = \{5\}$
$\varepsilon - CLOSURE(6) = \{6\}$

Nyní rozpitváme předpis pro odstranění epsilon přechodů:

$$\delta'(q,a) = \bigcup_{p \in \varepsilon - CLOSURE(q)} \delta(p,a).$$

Výraz výše znamená, že pokud chceme určit přechod ze stavu q pro vstupní symbol a nového automatu bez epsilon přechodů, vezmeme všechny stavy z epsilonového uzávěru stavu q z původního automatu, a přechody ze všech těchto stavů pro jeden vstupní symbol a sjednotíme do jednoho a zapíšeme do daného políčka tabulky.

Pro náš automat tak pro stav 0 sjednotíme řádky 0 a 4, pro stav 1 řádky 1 a 5, atd. Automat bez epsilon přechodů tedy vypadá následovně:

	a	b
→ 0	{0,1}	{0,4,5}
1	{4,5,6}	{2}
← 2	{3,6}	{5,6}
← 3	{3}	{3}
4	{}	{5}
5	{6}	{}
← 6	{6}	{6}

Druhý bod algoritmu stanoví, jak se určí koncové stavy.

$$F' = \{q : \varepsilon - CLOSURE(q) \cap F \neq \emptyset, q \in Q\}$$

Stav 2 se tedy stal koncovým, protože $\epsilon - CLOSURE(2) = \{2, 6\}$ a 6 byl koncový v původním automatu.

1.2 Determinizace

Determinizace de facto simuluje, co by se stalo, kdybychom procházeli včechny možné "cesty" automatem najednou. Takže jmenovka stavu vlastně obsahuje, kde všude by se automat po přečtení daného vstupu nacházel.

Začneme startovním stavem, u nějž přechodovou funkci akorát opíšeme (na začátku se nacházíme pouze v jednom stavu, takže přechodová funkce je identická), místo zápisu množiny však použijeme hranatých závorek. Vyznačíme tak, že se spíš jedná o jeden stav a ne množinu. Oba cílové stavy $[0, 1]$ a $[0, 4, 5]$ jsou nyní neoznačené, stav 0 je označený (už jsme jej zpracovali).

δ	a	b
→ $[0]$	$[0, 1]$	$[0, 4, 5]$

Pokračujeme dalším neoznačeným stavem, $[0, 1]$. Pro určení přechodové funkce pro tento stav nám stačí nakombinovat řádky pro stavy 0 a 1 z původního automatu, to nám říká definice ve slajdech. Platilo, že $\delta_{nfa}(0, a) = \{0, 1\}$ a $\delta_{nfa}(1, a) = \{4, 5, 6\}$, výsledný cílový stav je tedy $[0, 1, 4, 5, 6]$. Obdobnou operaci provedeme pro vstupní symbol b.

δ	a	b
$[0, 1]$	$[0, 1, 4, 5, 6]$	$[0, 2, 4, 5]$

Nezapomeňte, že "jmenovka" nového stavu je definována pomocí operace sjednocení, takže případné opakující se podstavy vynecháváme. Stejným způsobem určíme hodnotu přechodové funkce pro vstupní symbol b. Stav $[0, 1]$ je nyní označený, nové dva stavy neoznačené a pokračujeme dalším neoznačeným stavem. Časem začne naše přechodová funkce vyrábět stejné stavy (se stejnou jmenovkou), to je naprosto v pořádku, jinak bychom nikdy neskončili, navíc nových stavů může být jen $2^{|Q_{nfa}|}$ (množina podmnožin z Q). Pod jmenovkami stavů musíme pořád chápat množiny, takže stavy s různým pořadím stejných "podstavů" jsou identické.

Koncové stavy nového automatu jsou všechny takové, které obsahují aspoň jeden koncový stav z výchozího automatu.

	δ	a	b
$\rightarrow$	$[0]$	$[0,1]$	$[0,4,5]$
	$[0,1]$	$[0,1,4,5,6]$	$[0,2,4,5]$
	$[0,4,5]$	$[0,1,6]$	$[0,4,5]$
$\leftarrow$	$[0,1,4,5,6]$	$[0,1,4,5,6]$	$[0,2,4,5,6]$
$\leftarrow$	$[0,2,4,5]$	$[0,1,3,6]$	$[0,4,5,6]$
$\leftarrow$	$[0,1,6]$	$[0,1,4,5,6]$	$[0,2,4,5,6]$
$\leftarrow$	$[0,2,4,5,6]$	$[0,1,3,6]$	$[0,4,5,6]$
$\leftarrow$	$[0,1,3,6]$	$[0,1,3,4,5,6]$	$[0,2,3,4,5,6]$
$\leftarrow$	$[0,4,5,6]$	$[0,1,6]$	$[0,4,5,6]$
$\leftarrow$	$[0,1,3,4,5,6]$	$[0,1,3,4,5,6]$	$[0,2,3,4,5,6]$
$\leftarrow$	$[0,2,3,4,5,6]$	$[0,1,3,6]$	$[0,3,4,5,6]$
$\leftarrow$	$[0,3,4,5,6]$	$[0,1,3,6]$	$[0,3,4,5,6]$

Po determinizaci vznikne automat se spoustou koncových stavů (po menší úvaze zjistíme, že je "vyrobily" smyčky ve stavech 3 a 6), které jsou ale ekvivalentní. Ale musíme to dokázat tím, že se pokusíme automat minimalizovat.

1.3 Minimalizace

Minimalizace je založena na vlastnosti, které se říká ekvivalence stavů. Stavy jsou ekvivalentní, pokud se pro libovolné slovo chovají stejně, tedy shodně dorazí do koncového, nebo nekoncového stavu.

Tzn. pro ekvivalentní stavy p, q a libovolné $w \in T^*$ platí něco jako

$$\delta^*(q,w) \in F \Leftrightarrow \delta^*(p,w) \in F.$$

Pro limitní případ $w = \varepsilon$ musí být stavy shodně koncové nebo nekoncové. Funkci δ^* se říká rozšířená přechodová funkce, a lze ji definovat induktivně nějak takto:

$$\delta^*(q,\varepsilon) = q$$
$$\delta^*(q,wa) = \delta(\delta^*(q,w),a),$$

kde $q \in Q, w \in T^*, a \in T$.

Postupem uvedeným v přednášce (ač to není moc vidět) vlastně kontrolujeme tuto vlastnost iterativně od nejkratších řetězců. Začínáme s předpokladem, že stavy jsou ekvivalentní pro $w = \varepsilon$, tedy rozdělíme stavy do dvou tříd ekvivalence jen podle toho, zda jsou koncové či nikoliv.

$$I = \{\ [0],[0,1],[0,4,5]\ \}$$
$$II = \{\ [0,1,4,5,6],[0,2,4,5],[0,1,6],[0,2,4,5,6],[0,1,3,6],[0,4,5,6],$$
$$[0,1,3,4,5,6],[0,2,3,4,5,6],[0,3,4,5,6]\ \}$$

Potom přechodovou funkci pro každý stav vyplníme podle toho, do jaké skupiny daný přechod vede:

	I	a	b
$\rightarrow$	$[0]$	I	I
	$[0,1]$	II	II
	$[0,4,5]$	II	I

	$II.$	a	b
$\leftarrow$	$[0,1,4,5,6]$	II	II
$\leftarrow$	$[0,2,4,5]$	II	II
$\leftarrow$	$[0,1,6]$	II	II
$\leftarrow$	$[0,2,4,5,6]$	II	II
$\leftarrow$	$[0,1,3,6]$	II	II
$\leftarrow$	$[0,4,5,6]$	II	II
$\leftarrow$	$[0,1,3,4,5,6]$	II	II
$\leftarrow$	$[0,2,3,4,5,6]$	II	II
$\leftarrow$	$[0,3,4,5,6]$	II	II

Vidíme, že první skupinu musíme roztrhnout na tři a znovu přepsat přechodové funkce

I	a	b
$\rightarrow [0]$	III	IV

III	a	b
$[0,1]$	II	II

IV	a	b
$[0,4,5]$	II	IV

Druhá skupina se nezmění. Pokud budeme pokračovat, zjistíme, že skupina II se již nedá roztrhnout, všechny přechody totiž vedou do ní samé.

Iterace se zastaví v momentě kdy jsme nic nerozdělili. Výsledný automat bude mít stavy podle počtu výsledných skupin a přechodovou funkci spočtenou podle původního automatu a příslušnosti původních stavů do výsledných skupin.

	δ	a	b
$\rightarrow$	I	III	IV
$\leftarrow$	II	II	II
	III	II	II
	IV	II	IV

1.4 Časté problémy

Informace získané po stovce opravených zkouškových písemek a hodinách strávených nad opravou domácích úkolů

1. ε-přechody. Pamatujte, že funkce ε-CLOSURE je uzávěr. Jistě si vzpomenete na tranzitivní uzávěry binárních relací. Pro dosažení nějakého stavu za pomoci epsilon přechodů je možno použít více přechodů po sobě. Stejně tedy tak epsilon uzávěr musí být iterován tak dlouho, dokud to jde.
2. Odstraňování ε-přechodů. Obecně existují dva různé algoritmy. Druhý postup odstraňuje epsilon přechody v rámci determinizace a nebyl záměrně prezentován. Loni jsem to udělal a výsledkem bylo to, že jej studenti kombinovali s prvním, což vedlo na špatná řešení. Pokud vás i tak zajímá najdete jej všude na internetu, případně na eduxu na mé stránce v sekci vyučující, kde rozebírám rozdíly mezi oběma metodami.
3. Determinizace. Někteří opisují všechny stavy z nedeterministického automatu do deterministického. Jenže tyto stavy se pak nikde nevyskytují na pravé straně, tudíž jsou nedosažitelné a jen zabírají místo, nemluvě o minimalizaci.
4. Minimalizace. Skupiny stavů lze pouze rozdělovat, nikoliv slučovat. To, že stavy rozdělíte v nějaké iteraci do skupin znamená, že jste ukázali, že pro nějaké vstupní slovo stavy ekvivalentní nejsou. Už vůbec nemůžete prohlásit za ekvivalentní stavy koncový s nekoncovým. Takže jednoduché pravidlo: Rozdělovat, ale nikdy už neslučovat zpět.

5. Minimalizace 2. Předčasné ukonční minimalizace. To, že v některé iteraci jednu skupinu neroztrhneme, neznamená, že ji neroztrhneme příště. Může to být totiž ovlivněno roztržením jiné skupiny. Algoritmus ukončujeme opravdu tehdy, pokud se nezmění v jednom kroku vůbec nic...

1.5 Konečné automaty - příklady

1.5.1 Příklad

Sestrojte konečný automat nad abecedou $A = \{a,b\}$, který přijme slovo "abba".

Tohle je celkem jednoduché, pro všechno ostatní automat "spadne":

δ	a	b
$\rightarrow 0$	1	-
1	-	2
2	-	3
3	4	-
$\leftarrow 4$	-	-

1.5.2 Příklad

Sestrojte konečný automat nad abecedou $A = \{a,b\}$, který přijme slovo, které končí na "abba".

Tady si snadno pomůžeme nedeterminismem. Automatu umožníme před přečtením slova "abba" přečíst libovolnou posloupnost znaků jednoduchou smyčkou ve stavu 0.

δ	a	b
$\rightarrow 0$	$0,1$	0
1	-	2
2	-	3
3	4	-
$\leftarrow 4$	-	-

Zkuste si daný automat zdeterminizovat, mělo by vám vyjít (automat je minimální, ověřte si pokusem o minimalizaci):

δ	a	b
$\rightarrow [0]$	$[0,1]$	$[0]$
$[0,1]$	$[0,1]$	$[0,2]$
$[0,2]$	$[0,1]$	$[0,3]$
$[0,3]$	$[0,1,4]$	$[0]$
$\leftarrow [0,1,4]$	$[0,1]$	$[0,2]$

1.5.3 Příklad

Sestrojte konečný automat nad abecedou $A = \{a,b\}$, který přijme slovo, které začíná na "abba".

Zde je přístup opačný, smyčku umístíme na konec automatu

δ	a	b
$\rightarrow 0$	1	-
1	-	2
2	-	3
3	4	-
$\leftarrow 4$	4	4

Automat je deterministický

1.5.4 Příklad

Sestrojte konečný automat nad abecedou $A = \{a,b\}$, který přijme slovo, které obsahuje "abba".

Zkombinujeme oba přístupy:

δ	a	b
$\rightarrow 0$	$0,1$	0
1	-	2
2	-	3
3	4	-
$\leftarrow 4$	4	4

Po determinizaci:

δ	a	b
$\rightarrow [0]$	$[0,1]$	$[0]$
$[0,1]$	$[0,1]$	$[0,2]$
$[0,2]$	$[0,1]$	$[0,3]$
$[0,3]$	$[0,1,4]$	$[0]$
$\leftarrow [0,1,4]$	$[0,1,4]$	$[0,2,4]$
$\leftarrow [0,2,4]$	$[0,1,4]$	$[0,3,4]$
$\leftarrow [0,3,4]$	$[0,1,4]$	$[0,4]$
$\leftarrow [0,4]$	$[0,1,4]$	$[0,4]$

Automat zminimalizujeme:

δ	a	b
$\rightarrow [0]$	$[0,1]$	$[0]$
$[0,1]$	$[0,1]$	$[0,2]$
$[0,2]$	$[0,1]$	$[0,3]$
$[0,3]$	$[0,1,4]$	$[0]$
$\leftarrow [0,1,4]$	$[0,1,4]$	$[0,1,4]$

1.5.5 Příklad

Sestrojte konečný automat nad abecedou $A = \{a,b\}$, který přijme slovo "abba" a všechny jeho neprázdné předpony (prefix automata)

Na předpony existuje jednoduchý trik, automat bude mít více koncových stavů:

δ	a	b
$\rightarrow 0$	1	-
$\leftarrow 1$	-	2
$\leftarrow 2$	-	3
$\leftarrow 3$	4	-
$\leftarrow 4$	-	-

Automat je deterministický.

1.5.6 Příklad

Sestrojte konečný automat nad abecedou $A = \{a,b\}$, který přijme slovo "abba" a všechny jeho neprázdné přípony (suffix automata)

Na přijímání přípon si vypomůžeme epsilon přechodem, abychom mohli přeskočit libovolnou část řetězce. Do stavu 4 neskáčeme, chceme pouze neprázdné přípony.

δ	a	b	ϵ
$\rightarrow 0$	1	-	$1,2,3$
1	-	2	-
2	-	3	-
3	4	-	-
$\leftarrow 4$	-	-	-

Determinizovaný:

δ	a	b
$\rightarrow [0,1,2,3]$	$[1,4]$	$[2,3]$
$\leftarrow [1,4]$	-	$[2]$
$[2,3]$	$[4]$	$[3]$
$[2]$	-	$[3]$
$\leftarrow [4]$	-	-
$[3]$	$[4]$	-

1.5.7 Příklad

Sestrojte konečný automat nad abecedou $A = \{a,b\}$, který přijme slovo "abba" a všechny jeho souvislé podřetězce (factor automata).

Výsledek vznikne kombinací obou předchozích přístupů. Automat, který by přijímal pouze neprázdné podřetězce by byl o mnoho větší (zkuste si jak by vypadal ... je to vlastně sjednocení více automatů z bodu 1.5.5 - předpony ze slov abba, bba, ba, a), nám bude stačit tato zkratka, která přijímá i prázdné slovo.

δ	a	b	ϵ
$\rightarrow 0$	1	-	$1,2,3$
$\leftarrow 1$	-	2	-
$\leftarrow 2$	-	3	-
$\leftarrow 3$	4	-	-
$\leftarrow 4$	-	-	-

Deterministický:

δ	a	b
$\rightarrow \leftarrow [0,1,2,3]$	$[1,4]$	$[2,3]$
$\leftarrow [1,4]$	-	$[2]$
$\leftarrow [2,3]$	$[4]$	$[3]$
$\leftarrow [2]$	-	$[3]$
$\leftarrow [4]$	-	-
$\leftarrow [3]$	$[4]$	-

1.5.8 Příklad

Sestrojte konečný automat nad abecedou $A = \{0,1\}$, který přijme slova, která odpovídají binárnímu zápisu celých čísel dělitelných třemi.

Zdánlivě složitý úkol je vlastně jednoduchý a princip se podobá dělení se zbytkem, jaké známe ze základní školy. Mezi desítkovou soustavou a libovolnou jinou soustavou je pramalý rozdíl. Princip onoho dělení spočíval v postupném přibíráním číslic k prozatím vypočtenému výsledku. Např.

$42 : 2 = 21$, zbytek 1

```
101011 : 10 = 010101
1
10
 0
```

```
01
010
  0
  01
  011
    1
```

Konečný automat vlastně jen sleduje, co se děje průběžně se zbytkem. Každým sepsáním další číslice se zbytek v případě dvojkové soustavy vynásobí dvěma (shift doleva) a přičte se hodnota připsané číslice a poté vydělí dělitelem. Každý stav automatu tedy odpovídá jedné možné hodnotě zbytku a stav pro 0 bude koncový. Vstup automatu odpovídá "sepisované" číslici. Tedy například stav 2 odpovídá zbytku 2 a připsáním jedničky dostaneme $2 * 2 + 1 = 5$. To po dělení 3 dává opět 2.

δ	0	1
$\rightarrow \leftarrow 0$	0	1
1	2	0
2	1	2

1.5.9 Další příklady k procvičení

1. Sestrojte automat nad abecedou A = {0,1}, který přijme všechna slova, kde je počet jedniček dělitelný třemi.
2. Sestrojte automat nad abecedou A = {0,1}, který přijme všechna slova, kde je počet jedniček dělitelný dvěmi nebo pěti. Použijte metodu na skládání automatu (sjednocení jazyků).
3. Sestrojte automat nad abecedou A = {0,1}, který přijme všechna slova, kde je počet jedniček dělitelný dvěmi a zároveň pěti. Použijte metodu na skládání automatu (průnik jazyků).
4. Sestrojte automat nad abecedou A = {0,1}, který přijme všechna slova, která obsahují 000 a zároveň končí na 01.

2 Převody RG ↔ KA ↔ RV

2.1 Převod regulární gramatiky na konečný automat

Převod je celkem zřejmý. Pro každý neterminál vytvoříme stav automatu. Pro každé pravidlo ve tvaru $A \rightarrow aB$ máme přechod ze stavu A do stavu B pro vstupní symbol a. Pro terminální pravidla, tj. ve tvaru $A \rightarrow b$ vytvoříme přechod ze stavu A do speciálního nového stavu, označme jej třeba F, pro vstup b. Tento nový stav F bude koncový. Pokud je součástí gramatiky speciální pravidlo $S \rightarrow \varepsilon$, kde S je počáteční neterminál gramatiky, stav S našeho automatu bude také koncový (aby automat mohl přijmout prázdný symbol).

Příklad. Mějme gramatiku $G = (\{A, B, C\}, \{a, b, c\}, A, P\}$, kde množina P pravidel vypadá následovně:

$A \rightarrow \varepsilon \mid bB \mid a$
$B \rightarrow \ bB \mid aC \mid c$
$C \rightarrow \ bB \mid b \mid aA \mid aC$

Automat tedy bude vypadat asi takto:

δ	a	b	c
$\leftarrow \rightarrow A$	F	B	-
B	C	B	F
C	A, C	B, F	-
$\leftarrow F$	-	-	-

Koncové stavy budou dva. Stav A vznikl z počátečního symbolu gramatiky a v pravidlech máme $A \to \varepsilon$ (gramatika generuje prázdný řetěz).

2.2 Převod automatu na regulární gramatiku

Převod zpět je analogický předchozímu postupu. Zcela automaticky převedeme přechod z A do B pro vstup c jako pravidlo gramatiky $A \to cB$. Pokud je zároveň stav B koncový, je nutno přidat druhou variantu pravidla, a to $A \to c$.

Trochu problematické je to s počátečním stavem. Z něj sice snadno vytvoříme počáteční neterminál gramatiky, ovšem trable nastávají v případě, že je to zároveň stav koncový.

Reverzně k příkladu výše bychom napsali pravidlo $S \to \varepsilon$, to si ovšem můžeme dovolit (s odkazem na definici regulární gramatiky) jen v případě, že pak S nebude na pravé straně žádného pravidla (tj. do stavu S původního automatu "nic nevede"). Pokud se nám S na pravé straně nějakého pravidla přeci jen objeví (tj. v automatu do stavu S vede nějaký přechod), je nutno si pomoci trikem z přednášky a restriktivní podmínku v definici regulární gramatiky obejít.

Vytvoříme nový počáteřní neterminál R a pro něj pravidlo $R \to \varepsilon$. Dále na pravou stranu pravidla pro R okopírujeme všechny pravé strany pro neterminál S. Neterminál R pak splňuje podmínku pro regulární gramatiku. Může být přepsán na epsilon a zároveň se nevyskytuje nikde napravo.

Příklad. Mějme automat

δ	a	b	c
$\leftarrow \to A$	A	B	C
B	C	B	A
$\leftarrow C$	A, C	B	-

Naivně bychom pravidla gramatiky sestrojili následovně:

$A \to aA \mid a \mid bB \mid cC \mid c \mid \varepsilon$
$B \to aC \mid a \mid bB \mid cA \mid c$
$C \to aA \mid aC \mid a \mid bB$

Pravidlo $A \to \varepsilon$ odpovídá počátečnímu stavu, který je zároveň koncový, ale s ním má však definice regulární gramatiky z přednášky problém. Proto jej nahradíme trikem a zavedeme nový neterminál, který bude počátečním symbolem gramatiky a bude mít následující pravidla:

$R \to \varepsilon \mid aA \mid a \mid bB \mid cC \mid c$

Vlastně jsme zkopírovali pravidla pro A, a z něj teď pravidlo pro epsilon vypustíme. Výsledná gramatika tedy bude $G_1 = (\{R, A, B, C\}, \{a, b, c\}, R, P)$, kde P je sada pravidel:
$R \to aA \mid a \mid bB \mid cC \mid c \mid \varepsilon$
$A \to aA \mid a \mid bB \mid cC \mid c$
$B \to aC \mid a \mid bB \mid cA \mid c$
$C \to aA \mid aC \mid a \mid bB$

2.3 Převod regulární gramatiky na regulární výraz

Asi nejrychlejší metodou je použití regulárních rovnic. Další možností je postupná eliminace neterminálů, která analogicky odpovídá eliminaci stavů ekvivalentního automatu. Tento algoritmus bude na automatech popsán později.

Mějme gramatiku $G_2 = (\{A, B\}, \{a, b\}, A, P)$, kde P:

$A \to aA \mid bB \mid a$
$B \to aA \mid aB \mid bB \mid b$

Lusknutím prstu převedeme gramatiku na soustavu regulárních rovnic:

$A = aA + bB + a$
$B = aA + aB + bB + b$

Řešíme obdobně jako na základní škole soustavu N rovnic o N neznámých – dosazováním. Tady ovšem máme jinou sadu ekvivalentních úprav. Využijeme triku, že rovnici ve tvaru $A = \alpha A + \beta$ lze převést na výraz $A = \alpha^*\beta$.

Jelikož chceme ekvivalentní výraz k dané gramatice, výsledkem je výraz pro proměnnou odpovídající počátečnímu neterminálu gramatiky, tedy A. Zbavíme se tedy nejprve B a to tak, že z druhé rovnice vyjádříme B a dosadíme do rovnice č. 1.

Druhou rovnici si nejprve trochu ozávorkujeme a vytkneme B (pozor napravo, zřetězení není komutativní operace):
$B = (a + b)B + (aA + b)$.

Nyní druhou rovnici máme ve tvaru, který potřebujeme. Vyjádříme B:
$B = (a + b)^*(aA + b)$ (pozor "zbytek" bez B se přilepuje pomocí zřetězení, není tam +, to je častý zdroj chyb).

Dosadíme do první rovnice:
$A = aA + b(a + b)^*(aA + b) + a$

Roznásobíme druhý člen:
$A = aA + b(a + b)^*aA + b(a + b)^*b + a$

Vytkneme A vpravo za závorku:
$A = (a + b(a + b)^*a)A + b(a + b)^*b + a$

Vyjádříme A pomocí triku výše:
$A = (a + b(a + b)^*a)^*(b(a + b)^*b + a)$,
což je řešením.

POZOR. Při úpravách rovnic zapomeňte na roznásobování součtem zabaleným v iteraci. Rozhodně nelze roznásobit výraz $a(b + c)^*$. Na úpravy regulárních výrazů existuje kuchařka, kterou najdete ve slajdech.

Stejnou metodu řešení regulárních rovnic lze použít na převod automatu na regulární výraz. To buď nepřímou transformaci přes regulární gramatiku, nebo přímým generováním soustavy rovnic ze zdrojového automatu. Zde existují dvě varianty, pro příchozí nebo pro odchozí hrany automatu.

2.4 Převod regulárního výrazu na automat

2.4.1 Metoda sousedů

Mějme regulární výraz $V = (ab^* + b)^*ab^* + a^*$. Nejprve si všechny elementy regulárního výrazu očíslujeme.
$V = (a_1b_2^* + b_3)^*a_4b_5^* + a_6^*$.

Sestavíme si množinu počátečních symbolů Z, tedy očíslovaných symbolů, které se mohou vyskytovat na začátku nějakého slova vygenerovaného regulárním výrazem. Určite tam bude a_1 a b_3, díky tomu, že je první část v iteraci, může být na začátku i a_4. Díky + na nejvyšší úrovni i a_6. Tedy $Z = \{a_1, b_3, a_4, a_6\}$.

Podobně zkonstruujeme množinu koncových symbolů, tedy těch, které se mohou vyskytovat na konci nějakého vygenerovaného řetězce. $F = \{a_4, b_5, a_6\}$. a_4 se na konec dostane kvůli iteraci u členu b_5.

Teď zbývají sousedi, nejzrádnější jsou na nich iterace v kombinaci se znamínkem + uvnitř. Je potřeba postupovat systematicky.

$P = \{$
$a_1a_1, a_1b_2, a_1b_3, a_1a_4,$
$b_2a_1, b_2b_2, b_2b_3, b_2a_4,$
$b_3a_1, b_3b_3, b_3a_4,$
$a_4b_5,$
$b_5b_5,$

a_6a_6,
}.

Automat pak sestavujeme snadno. Pro každý očíslovaný symbol budeme mít stav automatu. Navíc přidáme nový počáteční stav, označme jej třeba q_0.

Začneme zpracováním množiny Z. Z počátečního stavu q_0 natahneme přechod pro každý člen množiny Z. Pro každý pár z množiny P pak vytvoříme přechod automatu. Vstupní symbol přechodu vždy odpovídá cílovému stavu (automat je homogenní).

Koncové jsou všechny stavy vyjmenované v množině F. Dále, v tomto případě koncový bude i stav q_0, protože regulární výraz generuje i prázdný řetěz.

δ	a	b
$\rightarrow \leftarrow q_0$	a_1, a_4, a_6	b_3
a_1	$a1, a4$	b_2, b_3
b_2	$a1, a4$	b_2, b_3
b_3	$a1, a4$	b_3
$\leftarrow a_4$	-	b_5
$\leftarrow b_5$	-	b_5
$\leftarrow a_6$	a_6	-

2.5 Převod automatu na regulární výraz

První možností je použít metody přes řešení soustavy regulárních rovnic. Automat pomocí metody pro příchozí nebo odchozí hrany převedeme na soustavu rovnic a řešíme pro počáteční stav (metoda odchozích hran) nebo pro součet koncových stavů (metoda příchozích hran).

Druhou možností je postupná eliminace stavů.

V tomto případě je nutno zavést rozšířené ohodnocení přechodů, kde je každý přechod automatu možno ohodnotit regulárním výrazem.

Postup je ilustrovaný v příloze (obrázek na stejném místě na eduxu). Nejprve je nutno si počáteční stav a koncové stavy "vytáhnout" do nových stavů S a F. Pak sekvenčně odstraňujeme stavy. Platí, že pokud máme stavy A,B,C a chceme odstranit stav B, tak místo něj vytvoříme přechod s ohodnocením $\alpha\beta^*\gamma + \delta$, kde α je ohodnocení přechodu z A do B, β je smyčka ve stavu B, γ je ohodnocení přechodu z B do C, δ je ohodnocení přechodu z A do C přímo. Pokud některý z přechodů neexistuje, dosadíme za patřičnou část prázdný jazyk a chováme se podle toho. Pokud neexistuje smyčka v B, jen ji vynecháme a výsledek bude $\alpha\gamma + \delta$. Pokud například neexistuje přechod z B do C, cesta "oklikou" přes B nelze použít a zůstane jen přechod z A do C přímo, tj. δ. Skončíme, až máme jen stavy S a F a jeden přechod mezi nimi, který je ohodnocen výsledným regulárním výrazem.

3 Klasifikace jazyků

Pokud po vás někdo bude chtít zařazení jazyka, je nutno si vzpomenout, že jednotlivé třídy Chomského klasifikace jsou podmnožinou té vyšší, tj. regulární jazyk je zároveň bezkontextový, kontextový i rekurzivně spočetný, totéž platí o třídách gramatik. Proto jsou často zadání na zařazení jazyků (nebo gramatiky) formulována tak, že se po vás chce buď:

1. Stanovit nejnižší možnou třídu jazyka (gramatiky), nebo
2. všechny třídy jazyka (gramatiky), tj. včetně odpovídajících nadmnožin.

V případě gramatik je klasifikace snadná, stačí podle tvaru pravidel určit třídu, do které gramatika spadá a podle znění otázky odpovědět správnou odpovědí (ad 1) nebo odpovědmi (ad 2).

3.1 Zařazení jazyka

V případě jazyka je to trochu více práce. Neexistují univerzální vzorečky, na jejichž konci nám vypadne cílová třída, ale musíme příslušnost k dané kategorii odvodit. Správnou třídu jazyka musíme ohraničit shora i zdola.

Shora ji omezíme sestrojením gramatiky, která jazyk generuje, případně automatem, který jazyk přijímá. Pak víme, že je jazyk je nejhůře tak složitý jako námi vytvořená gramatika/automat. Ovšem nemáme jistotu, zdali nepatří ještě do nižší kategorie. Například pokud sestrojíme bezkontextovou gramatiku pro nějaký jazyk, není jazyk nutně nejlépe bezkontextový, ale může být i regulární. Jen se nám regulární gramatiku nepodařilo sestrojit (nebo nás ji nenapadlo sestrojit). Proto je nutno třídu jazyka omezit i zdola, a to důkazem, že jazyk do nižší kategorie patřit nemůže. Tzn. u bezkontextového jazyka musíme ukázat, že není regulární (nemůžu sestrojit regulární gramatiku/automat), u bezkontextového zase, že nemůže být bezkontextový ani regulární, atd. Na to se nejlépe hodí Pumping lemma (pro regulární, případně bezkontextové jazyky) nebo Nerodova věta (pro regulární jazyky)

Jelikož jsme na vás hodní, a jelikož se vyšší třídy Chomského hierarchie moc neprobírají, ve zkouškových písemkách se objevují příklady na rozlišení mezi regulárním a bezkontextovým jazykem.

3.1.1 Regulární jazyk

Důkaz, že daný jazyk je regulární, je relativně snadný. Stačí sestrojit regulární gramatiku, která jazyk generuje, konečný automat, který jazyk přijímá, nebo levou kongruenci splňující Nerodovu větu (což je vlastně ten automat, akorát jinak popsaný). Zdola je zařazení jazyka omezeno faktem, že "nic jednoduššího" už není. Shora jsme to dokázali sestrojenou gramatikou/automatem.

3.1.2 Bezkontextový jazyk

Pokud chceme dokázat, že je jazyk minimálně bezkontextový (tj. není zároveň regulární), musíme nejprve ukázat, že opravdu aspoň bezkontextový je (např. sestrojením bezkontextové gramatiky) a zároveň dokázat, že není zároveň regulární (Pumping lemmatem, Nerodovou větou).

3.1.3 Příklad

Rozhodněte, zda je jazyk $L = \{a^n b^n, n \geq 0\}$ *regulární nebo bezkontextový. Pokud je regulární, sestrojte regulární gramatiku, pokud je bezkontextový, dokažte, že není regulární a sestrojte bezkontextovou gramatiku, která jej generuje.*

Bohužel neexistuje univerzální pravidlo, které nám řekne - jazyk je/není regulární, a neřekne nám, jak neregularitu snadno dokázat. Nicméně u příkladů, které se loni objevily ve zkouškových písemkách, lze takto rozhodnout celkem snadno. Je nutno se zaměřit na podmínky, které svazují/nesvazují k sobě počty jednotlivých písmenek. V zadaném přikladu očividně závisí počet písmen a na počtu písmen b (nebo spíš naopak). Nad tím samotným se můžeme zamyslet. Musíme si nějakým způsobem zapamatovat, kolik jsme přečetli aček, abychom mohli dovolit jen ten samý počet bček. Jelikož konečný automat má jedinou paměť v podobě svých stavů, museli bychom rozlišit n různých stavů, což je v případě nekonečného $n \geq 0$ nemožné (konečný automat musí mít konečnou množinu stavů). Takto intuitivně bychom mohli chápat, že zde si s regulárním jazykem nevystačíme a budeme potřebovat něco složitějšího.

3.2 Pumping lemma (PL)

Nechť jazyk L je regulární. Pak platí (implikace), že existuje nějaká konstanta p, kde pro všechny řetězce w z jazyka L delší než p (tj. $w \in L, |w| \geq p$) existuje rozdělení $w = xyz$ takové, že platí:

1. $|xy| \leq p$ (část y je nejdále p od začátku slova)
2. $|y| \geq 1$ (část y není prázdná)

3. $\forall k \geq 0 : xy^k z \in L$ (částí y lze pumpovat a všechna vzniklá slova jsou v L).

Jelikož se jedná o implikaci, lemma můžeme použít pouze na důkaz toho, že jazyk regulární není (vyvrácením pravé strany implikace). Pokud si postupně znegujeme dané tvrzení, vyplývá nám, že musíme pro každou konstantu p najít slovo, pro které dané rozdělení, při kterém by šlo pumpovat, nelze sestrojit.

Dá se to tedy představit jako souboj dvou lidí:

- A: Jazyk L je regulární, konstanta p existuje.
- B: Tak zkus rozdělit tohle slovo: w
- A: Hm, tak w rozdělit sice umím hned několika způsoby, ale ani v jednom případě nemůžu libovolně pumpovat částí y. Jazyk tedy regulární není.

3.2.1 Příklad

Rozhodněte, zda je jazyk $L = \{a^n b^n, n \geq 0\}$ regulární nebo bezkontextový. Pokud je regulární, sestrojte regulární gramatiku, pokud je bezkontextový, dokažte, že není regulární a sestrojte bezkontextovou gramatiku, která jej generuje.

Pro navrhovanou konstantu p se nabízí zvolit slovo $a^p b^p$, které do jazyka L patří. Podle podmínek daných v definici PL musí být pumpovaná část "někde na začátku", tj. v sekci písmen a. Zároveň musí mít délku aspoň 1. Je tedy jasné že při pumpování dolů ($k = 0$) i nahoru ($k \geq 2$) se změní počet písmen a, který bude buď menší nebo větší než počet bček a pumpovaná slova tedy do jazyka L nepatří. Gramatika pro tento jazyk je velmi jednoduchá, stačí nám dvě pravidla $\{S \to aSb, \quad S \to \varepsilon\}$. Jazyk bude tedy bezkontextový. Regulární být nemůže, viz důkaz níže.

Formálně: Nechť existuje hodnota p, pro kterou PL platí. Pak zvolme slovo $w = a^p b^p$, $|w| > p$. Podle podmínek 1 a 2 PL můžeme rozdělit slovo na $w = xyz$ pouze tak, že $x = a^r$, $y = a^s; s \geq 1$ a nakonec $z = a^{p-r-s} b^p$. Podle podmínky 3 (pumpování) budou vznikat slova $\{a^r a^{ks} a^{p-r-s} b^p, k \geq 0\}$.

Porovnáním koeficientů zjistíme, že:

$$\begin{array}{rcl} r + ks + (p - r - s) & \neq & p \\ p + (k-1)s & \neq & p \end{array} \quad \text{pro libovolné } k \neq 1 \text{ a } s \geq 1.$$

Pumpovat "dolů" tedy nelze, platí, že $a^r a^{ks} a^{p-r-s} b^p \notin L$ pro $k = 0$ a PL není splněno. Obdobně, pumpovat "nahoru" nelze, platí, že $a^r a^{ks} a^{p-r-s} b^p \notin L$ pro $k \geq 2$ a PL není splněno.

Méně formální vysvětlení: Podle pravidel 1 a 2. z výše citovaného PL lze slovo rozdělit pouze tak, že část y bude obsahovat samá ačka, a to alespoň jedno. Pumpováním dolů ($k = 0$) bude platit $pocet(a) < pocet(b)$ a slovo nebude součástí jazyka. Obdobně, při pumpování nahoru ($k \geq 2$) bude platit $pocet(a) > pocet(b)$

3.2.2 Příklad

Zařaďte jazyk $\{a^m b^n; m > n \geq 0\}$.

Opět vidíme, že si musíme umět pamatovat počet aček, a zároveň zajistit, že bček bude méně.

Abychom rozbili PL, musíme pumpovat dolů v části s ačky. Pumpováním ačky nahoru nic nezmůžeme, generovaná slova do jazyka stále budou patřit. Musíme tedy zajistit, že se nezdaří pumpování dolů. Takže zvolíme slovo, u kterého to opravdu nejde. Tím je slovo $a^{p+1} b^p$, tedy slovo, ve kterém je nerovnost počtu znaků těsně na hranici platnosti podmínky. Při pumpování dolů, byť s y délky 1, se počet aček stane nežádoucím a slovo do jazyka patřit nebude.

Pokud bychom zvolili $a^{p+2} b^p$, náš soupeř může zvolit y délky 1 a pumpnout i dolů. Slovo $a^{p+1} b^p$ totiž do jazyka patří též a nic bychom nedokázali.

Formálně: Nechť existuje hodnota p, pro kterou PL platí. Pak zvolíme slovo $w = a^{p+1}b^p$, $|w| > p$. Podle podmínek 1 a 2 PL můžeme rozdělit slovo $w = xyz$ pouze tak, že $x = a^r$, $y = a^s; s \geq 1$ a nakonec $z = a^{p+1-r-s}b^p$. Podle podmínky 3 (pumpování) budou vznikat slova $\{a^r a^{ks} a^{p+1-r-s} b^p, k \geq 0\}$.

Porovnáním koeficientů zjistíme, že:

$$\begin{array}{rcl} r + ks + (p + 1 - r - s) & \leq & p \\ p + 1 + (k - 1)s & \leq & p \end{array} \quad \text{pro } k = 0 \text{ (a } s \geq 1).$$

Pumpovat "dolů" ($k = 0$) tedy nelze, tedy platí, že $a^r a^{ks} a^{p+1-r-s} b^p \notin L$ pro $k = 0$ a PL není splněno.

Neformálně: Podle pravidel 1 a 2. z výše citovaného PL lze slovo rozdělit pouze tak, že část y bude obsahovat samá ačka, a to alespoň jedno. Pumpovaním dolů ($k = 0$) bude platit $pocet(a) \leq pocet(b)$ a slovo nebude součástí jazyka.

Gramatika je podobná té z přechozího příkladu, jen musíme zajistit nerovnost, tzn. přidat alespoň jedno ačko navíc.

$S \to aSb, \quad S \to aX, \quad X \to aX \mid \varepsilon$

První pravidlo zajišťuje párování počtu a a b, druhé pravidlo se postará o nadbytek počtu aček aspoň o 1. Třetí pravidlo dovolí mít aček ještě víc (aby platila nerovnost). Takže jazyk je bezkontextový.

Vsuvka: Někdo by namítl, že by bylo snažší donutit soupeře pumpovat v části s bčky nahoru. Toto ale vzhledem k podmínkám 1 a 2 obecně nelze zajistit. Soupeř si může za y zvolit cokoliv ve vzdálenosti p od začátku slova a pumpovat klidně v části s ačky.

Je sice pravda, že v tomto příkladě by i u slova $a^{\lceil p/2 \rceil + 1} b^{\lceil p/2 \rceil}$ selhal v ačkách při pumpování dolů, v bčkách při pumpování nahoru (a on musí najít rozdělení, kde lze pumpovat nahoru i dolů současně), obecně je dobré na tuto možnost soupeře nezapomínat.

3.2.3 Příklad

Zařaďte jazyk $\{a^m b^n; n > m \geq 0\}$.

Zrcadlové obrácený příklad výše. Zde nám stačí soupeře přinutit v části s a pumpovat nahoru.

Formálně: Nechť existuje hodnota p, pro kterou PL platí. Pak zvolíme slovo $w = a^p b^{p+1}$, $|w| > p$. Podle podmínek 1 a 2 PL můžeme rozdělit slovo $w = xyz$ pouze tak, že $x = a^r$, $y = a^s; s \geq 1$ a nakonec $z = a^{p-r-s}b^{p+1}$. Podle podmínky 3 (pumpování) budou vznikat slova $\{a^r a^{ks} a^{p-r-s} b^{p+1}, k \geq 0\}$.

Porovnáním koeficientů zjistíme, že:

$$\begin{array}{rcl} r + ks + (p - r - s) & \geq & p + 1 \\ p + (k - 1)s & \geq & p + 1 \end{array} \quad \text{pro } k \geq 2 \text{ (a } s \geq 1).$$

Pumpovat "nahoru" ($k \geq 2$) tedy nelze, protože $a^r a^{ks} a^{p-r-s} b^{p+1} \notin L$ pro $k \geq 2$ a PL není splněno.

Neformálně: Podle pravidel 1 a 2. z výše citovaného PL lze slovo rozdělit pouze tak, že část y bude obsahovat samá ačka, a to alespoň jedno. Pumpovaním nahoru ($k = 2, 3, ...$) se počet aček zvýší oproti počtu bček a slovo nebude součástí jazyka.

Gramatika generuje navíc bčka, takže je až na lehkou modifikaci podobná té z předchozího příkladu.
$S \to aSb, \quad S \to bX, \quad X \to bX \mid \varepsilon$

3.2.4 Příklad

Zařad'te jazyk $L = \{a^n b^n, 3 \geq n \geq 0\}$

Toto je chyták. Na první pohled tam zůstává nutnost párování počtu *a*ček a *b*ček, nicméně jejich počet je omezený. Jazyky je tedy konečný, obsahuje pouze slova $L = \{\varepsilon, ab, aabb, aaabbb\}$. Konečné jazyky jsou zároveň regulární, lze totiž pro ně sestrojit regulární gramatiku nebo konečný automat, systematicky například metodou pro konstrukci automatu pro sjednocení jazyků (jednotlivých slov).

3.2.5 Příklad

Zařad'te jazyk $L = \{a^i b^j c^k d^l, j = l \geq 0, i \geq 1, k \geq 0\}$

Problém je podobný těm výše. Potřebujeme shodu počtu *b*ček a *d*ček, do toho se pletou písmena a a c, na jejich počet není kladen žádný vztah s ostatními. Jazyk jasně bude bezkontextový, gramatika by byla přibližně:

$S \to AB$
$A \to aA \mid a$
$B \to bBd \mid C$
$C \to cC \mid \varepsilon$

Nakreslete si derivační strom pro nějaké slovo z jazyka, všimněte si rozdílů v pravidlech pro neterminály A a C, znaků a je totiž nutno zajistit nenulový počet.

Důkaz, že jazyk není regulární, je nejsnadnější pomocí pumping lemmatu. Potřebujeme tvrzení regulárnosti rozbít na vztahu symbolů b a d. Jak bylo zmíněno výše, je vhodnější zaměřit se na levější z dvojice. Zároveň je nutno ostatní symboly nastavit tak, aby nám nepřekážely, nebo nám nedovolovaly slovo rozdělit tak, aby v nich šlo pumpovat. Teď jde především o symbol a. Aby se v části s *a*čky nedalo pumpovat, je nutné slovo zvolit uvážlivě, a to na hranici příslušnosti do jazyka.

Zvolíme proto slovo $ab^p d^p$. Počet *a*ček jsme zvolili nejmenší možný (1), počet *c*ček taktéž, tedy 0. Důvod je následovný. Musíme ukázat, že slovo nelze rozdělit tak, aby šlo pumpovat. V tomto případě to lze třemi způsoby:

1. Pumpovat budeme samotným *a*čkem. To nemůžeme směrem dolů, slovo s nulovým počtem *a*ček do jazyka nepatří. Z tohoto důvodu jsme zvolili právě takové slovo. Již počet *a*ček 2 by totiž dovolil pumpovat jen jedním, a to i směrem dolů.
2. Pumpovat budeme podřetězcem s *a*čkem na začátku a několika *b*čky. To nejde, pumpováním nahoru vznikne slovo s alternujícími symboly a a b.
3. Pumpovat budeme jen v sekci s *b*čky. To nemůžeme směrem nahoru ani dolů.

Jak je vidět, pokud chceme PL vyvrátit, musíme ukázat, že existuje slovo, které nelze rozdělit žádným způsobem. Na našem slovu $ab^p d^p$, které jsme zvolili jako protipříklad, jsme ukázali, že rozdělení nelze zvolit žádným způsobem. Tím jsme dokázali, že pravou stranu PL nelze splnit pro všechna slova delší než p. V písemce by to samozřejmě chtělo popsat formálněji.

3.2.6 Další příklady k procvičení

Zařad'te, dokažte a sestrojte příslušnou gramatiku. Časem sem možná dodám řešení.

- $\{ww^R, w \in \{a, b\}^*\}$ (R = reverzace, zrcadlově převráceno)
- $\{a^i b^j c^k d^l, i = j \geq 0, k = l \geq 0\}$
- $\{a^i b^j c^k d^l, i = l \geq 1, j = k \geq 1\}$