8. Regulární jazyky. Konečné automaty, regulární gramatiky a regulární výrazy. Minimalizace konečného automatu. Převod nedeterministického konečného automatu na deterministický automat. Vztah mezi konečnými automaty a regulárními jazyky. Použití pumping lemmatu pro regulární jazyky.

Jazykem se rozumí libovolná množina slov nad nějakou abecedou. Jazyky mohou být konečné i nekonečné.

Slovo je libovolná konečná posloupnost znaků z abecedy.

Abeceda je libovolná konečná množina, jejíž prvky se nazývají **znaky**.
Abeceda může být i prázdná množina, zapisuje se Σ.

- prázdné slovo ε, délka slova, počet výskytů znaku ve slově, množina všech slov Σ*, množina všech neprázdných slov Σ+, jazyky nad Σ jsou tedy podmnožiny Σ*
- operace nad slovy: zřetězení (.), podslovo (prefix, sufix), i-tá mocnina (na nultou = ε), zrcadlový obraz
- operace nad jazyky: sjednocení, průnik, rozdíl, zřetězení, i-tá mocnina (na nultou = {ε}), (pozitivní) iterace, doplněk (Σ* \ L), zrcadlový obraz
- třída jazyků je uzavřena na operaci, pokud výsledek operace patří do třídy jazyků

1.2.1 Pojem gramatiky

Definice 1.2. *Gramatika* $\mathcal{G}$ je čtveřice (N, Σ, P, S), kde

- N je neprázdná konečná množina *neterminálních symbolů* (stručněji: *neterminálů*).
- Σ je konečná množina *terminálních symbolů (terminálů)* taková, že $N \cap \Sigma = \emptyset$. Sjednocením N a Σ obdržíme množinu *všech symbolů* gramatiky, kterou obvykle označujeme symbolem V.
- $P \subseteq V^*NV^* \times V^*$ je konečná množina *pravidel*. Pravidlo (α, β) obvykle zapisujeme ve tvaru $\alpha \to \beta$ (a čteme jako „α přepiš na β“).
- $S \in N$ je speciální *počáteční* neterminál (nazývaný také *kořen gramatiky*).

Prvky množiny V^*, které lze odvodit z počátečního neterminálu, nazýváme *větnými formami* gramatiky $\mathcal{G}$. Přesněji, $\alpha \in V^*$ je větná forma právě když $S \Rightarrow^* \alpha$. Větná forma, která neobsahuje žádné neterminály, se nazývá *věta*. Množina všech vět tvoří jazyk *generovaný* gramatikou $\mathcal{G}$, označovaný jako $L(\mathcal{G})$:

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

Gramatiky $\mathcal{G}_1$ a $\mathcal{G}_2$ nazveme *jazykově ekvivalentní* (dále jen ekvivaletní), právě když generují tentýž jazyk, tj. $L(\mathcal{G}_1) = L(\mathcal{G}_2)$.

Chomského hierarchie gramatik (jazyků)

typ 0 Libovolná gramatika je gramatikou typu 0; na tvar pravidel se nekladou žádné omezující požadavky. Někdy též se takové gramatiky označují jako gramatiky bez omezení či frázové gramatiky (phrase grammars).

typ 1 Gramatika je typu 1 (nebo též *kontextová*[1], Context-Sensitive, CSG, méně často též *monotónní*), jestliže pro každé její pravidlo $\alpha \to \beta$ platí $|\alpha| \leq |\beta|$ s eventuelní výjimkou pravidla $S \to \varepsilon$, pokud se S nevyskytuje na pravé straně žádného pravidla.

typ 2 Gramatika je typu 2 (též *bezkontextová*, Context-Free, CFG), jestliže každé její pravidlo je tvaru $A \to \alpha$, kde $|\alpha| \geq 1$ s eventuelní výjimkou pravidla $S \to \varepsilon$, pokud se S nevyskytuje na pravé straně žádného pravidla.

typ 3 Gramatika je typu 3 (též *regulární* či *pravolineární*[2]), jestliže každé její pravidlo je tvaru $A \to aB$ nebo $A \to a$ s eventuelní výjimkou pravidla $S \to \varepsilon$, pokud se S nevyskytuje na pravé straně žádného pravidla. [3]

Hierarchie gramatik také určuje příslušnou hierarchii *jazyků*. Jazyk L je regulární (resp. bezkontextový, kontextový, typu 0) pokud existuje regulární (resp. bezkontextová, kontextová, typu 0) gramatika $\mathcal{G}$ taková, že $L(\mathcal{G}) = L$. Nyní je již zřejmý smysl „výjimky" týkající se pravidla $S \to \varepsilon$; kdybychom ji nepovolili, stal by se z libovolného (třeba i regulárního) jazyka po přidání prázdného slova jazyk typu 0, který nelze popsat ani kontextovou gramatikou. To by bylo značně nepřirozené – přidáním jediného slova se „charakter" obecně (a též i obvykle) nekonečného jazyka příliš nezmění.

Gramatikami sa nedajú popísať všetky triedy jazykov, pretože existujú aj jazyky ktoré nie sú ani typu 0. napr nad abecedou {a} existuje jazyk ktorý nie je typu 0. Dokaz spociva v tom, že množina všetkých slov nad abecedou {a} je spočítateľne nekonečná, čiže množina všetkých jazykov nad touto abecedou bude taktiež nespočítateľná. Zato jazykov typu 0 nad abecedou {a} je však iba spočítateľne mnoho, pretože každá konečná reprezentácia (popisný aparát), ktorou je aj gramatika, dokáže popísať najviac spočítateľnú množinu jazykov.

Konečný automat

Abstraktním modelem konečně stavových systémů jsou tzv. *konečné automaty*. Konečný automat je vybaven konečně stavovou řídící jednotkou (tj. konečnou pamětí), čtecí hlavou a páskou, na které je zapsané vstupní slovo – viz obrázek 2.1. Na začátku výpočtu je hlava umístěna na nejlevějším políčku pásky. Automat na základě přečteného symbolu a momentálního stavu svůj stav změní a posune čtecí hlavu o jedno políčko vpravo. Výpočet končí, pokud se automat „zablokuje", nebo přečte celé vstupní slovo. Slovo zapsané na pásce je automatem *akceptováno*, pokud je celé přečteno a výsledný stav je některý z předem určených *koncových* stavů. Množina všech slov, která daný konečný automat $\mathcal{M}$ akceptuje, tvoří jazyk akceptovaný automatem M.

Konečný automat je pětice *(Q, Σ, δ, q_0, F)*, kde

- *Q* – konečná neprázdná množina stavů
- *Σ* – konečná množina vstupních symbolů, vstupní abeceda.
- *δ* – parciální přechodová funkce *δ: Q x Σ→ Q*
- q_0 je prvkem *Q*, a je to tzv. *počáteční* stav.
- *F* je podmnožinou *Q*, a je to množina *koncových* stavů.

Rozšířená přechodová funkce je parciální funkce definovaná induktivně vzhledem k délce slova a určuje následující dvě situace:

Slovo akceptované automatem *M* je právě takové slovo, pod kterým automat přejde z počátečního stavu do koncového stavu.

Jazyk akceptovaný automatem *M* je množina *všech* slov, na kterých automat přejde z počátečního stavu do koncového stavu.

Pro dva automaty je možné sestrojit automat rozpoznávající průnik, sjednocení nebo rozdíl jazyků jimi rozpoznávanými – synchronní paralelní kompozice automatů. Také automat pro komplement jazyka. Podmínkou však jsou totální přechodové funkce.

Konečný automat možno reprezentovať popisom všetkých zložiek pätice z definície, pomocou tabuľky prechodovej funkcie, pomocou prechodového grafu, alebo výpočtovým (stavovým) stromom.

Lemma 2.6. *Ke každému FA $\mathcal{M}$ existuje ekvivalentní FA $\mathcal{M}'$ s totální přechodovou funkcí.*

Důkaz: Nechť $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$. Automat $\mathcal{M}'$ sestrojíme tak, že ke stavům automatu $\mathcal{M}$ přidáme nový nekoncový stav p a chybějící přechody do něj „nasměrujeme“. Tedy $\mathcal{M}' = (Q \cup \{p\}, \Sigma, \delta', q_0, F)$, kde $p \notin Q$ a δ' je definována takto:

$$\delta'(q, a) = \begin{cases} \delta(q, a) & \text{je-li } \delta(q, a) \text{ definováno,} \\ p & \text{jinak.} \end{cases}$$

(Totální) funkce z A do B je relace $f \subseteq A \times B$ kde pro každé $x \in A$ existuje *právě jedno* $y \in B$ takové, že $(x, y) \in f$.
$(x, y) \in f$ je ekvivalentní zápisu $f(x) = y$. A je definiční obor, B je obor hodnot.
Parciální funkce z množiny A do množiny B je relace $f \subseteq A \times B$ kde pro každé $x \in A$ existuje *nejvýše jedno* $y \in B$ takové, že $(x, y) \in f$.

Automat pro komplement

K automatu *M* = (*Q*, $^{\Sigma}$, $^{\delta}$, q_0, *F*) s *totální* **přechodovou funkcí** sestrojíme automat $^{\overline{M}}$ rozpoznávající jazyk *co–L*(*M*) jako $^{\overline{M}}$ = (*Q*, $^{\Sigma}$, $^{\delta}$, q_0, *Q* – *F*).

Pumping Lemma („o vkládání“)

Je to prostředek jak dokázat, že nějaký daný jazyk není regulární. Představuje nutnou podmínku, kterou musí každý regulární jazyk splňovat. Nutnou a postačujúcou podmienkou pre regularitu jazyka je tzv. Myhillova-Nerodova veta.

Pumping lemma říká, že v dostatečně dlouhém slově w, které patří do jistého jazyka lze nalézt tři části x, y a z, přičemž y může být i celé slovo. y lze pak libovolně opakovat či zcela vyloučit a výsledné slovo bude patřit do jazyka. Slovo musí být delší nebo rovno než pumpovací konstanta p, a část xy musí být kratší nebo rovno p.

Formálně řečeno, jestliže je jazyk L regulární, pak existuje číslo $p > 0$ takové, že každý řetězec w z L jehož délky alespoň p může být zapsán ve tvaru
$w = xyz$
kde x,y,z jsou taková, že $|xy| <= p$, $|y| > 0$ a xy^iz patří do L pro každý integer i.

Pro důkaz, že jazyk není regulární se používá negace pumping lemma.

- pro *libovolné* $n \in \mathbb{N}$ (pumpovací konstantu)
- vždy *existuje* takové slovo $w \in L$, které má délku alepoň n, a pro které platí, že
- při *libovolném* rozdělení slova w na takové tři části x, y, z, že $|xy| \leq n$ a $y \neq \varepsilon$
- vždy *existuje alespoň jedno* $i \in \mathbb{N}_0$ takové, že $xy^iz \notin L$.

Pak z PL plyne, že L není regulární.

Znovu si tedy uvědomme, že při použití PL k důkazu, že jazyk není regulární, volíme slovo w a počet pumpování i (viz výše podtržené existenční kvantifikátory). Nevolíme ani pumpovací konstantu n, ani rozdělení na podslova x, y, z.

Ideální je školní důkaz pro jazyk $L = \{a^n b^n : n \geq 0\}$. Hledání y v řetězcích tohoto jazyka se rozpadne na tři případy:

1. Samá a: Nelze iterovat, neboť výsledné slovo by nepatřilo do jazyka.
2. Samá b: Nelze iterovat, neboť výsledné slovo by nepatřilo do jazyka.
3. Řetězec $\{a\}^+.\{b\}^+$: Nelze ho iterovat, neboť by opět nepatřilo do jazyka.

Intuitivní důkaz tvrzení říká:

- Pro konečné jazyky se jako pumpovací konstanta vezme délka nejdelšího slova.
- Pro nekonečné jazyky musí existovat nějaký minimální konečný automat, který je akceptuje. Spočítá se počet jeho stavů a vezme se jako pumpovací konstanta p. Pokud automat akceptuje i slova delší než p, pak musí některými stavy procházet vícekrát. Jeden tento stav označíme S. Transakce, které automat posunou ze stavu S, a zpět do stavu S, akceptují nějaký řetězec. Tento řetězec je y v pumping lemmatu, tj. platí tvrzení lemma, že lze slovo nafukovat i že lze nafukování přeskočit rovnou dál.

Minimalizace konečného automatu

Konečné automaty nacházejí velmi široké uplatnění v technické praxi (viz část 2.4). Z hlediska efektivity a nákladnosti implementace je důležité, aby počet stavů byl pokud možno co nejmenší. Přirozeným problémem je proto konstrukce *minimálního* automatu (tj. automatu s nejmenším počtem stavů), který rozpoznává daný regulární jazyk L. V této části ukážeme, že minimální automat lze sestrojit poměrně jednoduchým způsobem — stačí mít k dispozici *nějaký* konečný automat, který rozpoznává L. Minimální automat pak obdržíme ztotožněním některých jeho stavů.

Minimální automat lze sestrojit z jakéhokoliv automatu rozponávající nějaký jazyk L. . Existence minimálního konečného automatu souvisí s Myhill–Nerodovou větou, kterou můžeme přeformulovat takto:

Počet stavů libovolného minimálního automatu rozpoznávajícího jazyk L je roven indexu prefixové ekvivalence $\sim_L$. (Takový konečný automat existuje právě když index $\sim_L$ je konečný.)

Důsledek M-N věty:

Minimální konečný automat akceptující jazyk L je určen jednoznačně až na isomorfismus (tj. přejmenování stavů).

Jistě nelze ztotožnit nějaký koncový stav p s nekoncovým stavem q. Pokud totiž $p = \hat{\delta}(q_0, x)$ a $q = \hat{\delta}(q_0, y)$, pak x musí být akceptován a y zamítnut, a to i po ztotožnění p a q. Není však způsob, jak zajistit, že „ztotožněný“ stav má někdy akceptovat a někdy zamítat. Dále, pokud bychom ztotožnili nějaké p a q, pak bychom měli ztotožnit i jejich následníky $\delta(p, a)$ a $\delta(q, a)$, abychom dodrželi funkcionalitu (tzv. determinismus) δ: pro daný stav a symbol je jednoznačně určen následník. Z těchto dvou úvah plyne, že nemůžeme ztotožnit p a q, pokud $\hat{\delta}(p, x) \in F$ a současně $\hat{\delta}(q, x) \notin F$ pro nějaké x. Ukazuje se, že tato podmínka je nutná i postačující pro rozhodování, kdy dva stavy ztotožnit, tj. pokud pro nějaké x $\hat{\delta}(p, x) \in F$ a současně $\hat{\delta}(q, x) \notin F$, pak stavy nemůžeme ztotožnit; pokud žádné takové x neexistuje, pak je ztotožnit můžeme. Tyto úvahy lze formalizovat takto:

Definice 2.32. Nechť $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ je FA bez nedosažitelných stavů, jehož přechodová funkce je totální. Pro každý stav q definujeme jazyk $L(q) \subseteq \Sigma^*$ předpisem

$$L(q) = \{x \in \Sigma^* \mid \hat{\delta}(q, x) \in F\} .$$

Stavy p, q nazveme *jazykově ekvivalentní*, psáno $p \equiv q$, pokud $L(p) = L(q)$, tj.

$$p \equiv q \iff \forall x \in \Sigma^* : (\hat{\delta}(p, x) \in F \iff \hat{\delta}(q, x) \in F)$$

Příklad:

Mějme automat *M* zadaný tabulkou.

M	a	b
→ 1	2	-
2	3	4
← 3	6	5
4	3	2
← 5	6	3
← 6	2	-
7	6	1

1. Provedeme odstranění nedosažitelných stavů, neboť do stavu 7 nevede žádná cesta.

M	a	b
→ 1	2	N
2	3	4
← 3	6	5
4	3	2
← 5	6	3
← 6	2	N
N	N	N

2. Rozdělíme stavy na koncové a nekoncové (stavy v relaci =0). Koncové stavy jsou v relaci ekvivalence s indexem 0.

	M	a	b
I	→ 1	I	I
	2	II	I
	4	II	II
	N	I	I
II	← 5	II	II
	← 6	I	I
	← 3	II	II

3. Sdruží se řádky, které mají sloupce vyplněné stejně. Opakujeme pořád dokola.
Rozdělujeme stavy podle ekvivalence zvyšováním i, dokud (=*i*) != (=*i*-+).

	M	a	b
I	→ 1	III	I
	N	I	I
III	4	IV	III
	2	IV	III
II	← 6	III	I
IV	← 5	II	IV
	← 3	II	IV

	M	a	b
I	→ 1	III	V
V	N	V	V
III	4	IV	III
	2	IV	III
II	← 6	III	V
IV	← 5	II	IV
	← 3	II	IV

4. Pokud dojde k tomu, že v každé skupině jsou řádky vyplněné stejně, není důvod co rozdělovat, relace je nalezena, nalezli jsme redukt.

	a	b
I	III	II
II	II	II
III	IV	III
V	III	II
IV	V	IV

Převod nedeterministického na deterministický

Nedeterministický konečný automat je zařízení, které je velmi podobné konečnému automatu z definice 2.1. Jediný rozdíl je v tom, že nedeterministický automat nemusí mít pro daný stav a vstupní symbol určen následující stav jednoznačně (na přechodových grafech si to lze představit tak, že z jednoho uzlu může vycházet více hran se stejným návěštím). Není tedy předem jasné, do jakého stavu se automat dostane po zpracování daného slova w, neboť automat si během výpočtu může „vybírat" jeden z možných následujících stavů. Slovo w bude akceptováno, pokud *alespoň jeden* z možných výpočtů nad slovem w skončí v koncovém stavu.

Formálne se teda přechodová funkce definuje jako totální zobrazení do množiny všech podmnožin stavů. $\delta : Q \times \Sigma \rightarrow 2^Q$.
Každý nedeterministický automat k sobě má nějaký ekvivalentní deterministický automat.

Navrhnout nějaký deterministický automat je obtížné, ale nedeterministický může být lehčí. Ten se dá následně algoritmicky převést na deterministický.

Algoritmus transformace NFA na DFA

Vstup: NFA $M = (Q, \Sigma, \delta, q_0, F)$.

Výstup: Ekvivalentní DFA $M' = (Q', \Sigma, \delta', \{q_0\}, F')$ bez nedosažitelných stavů a s totální přechodovou funkcí.

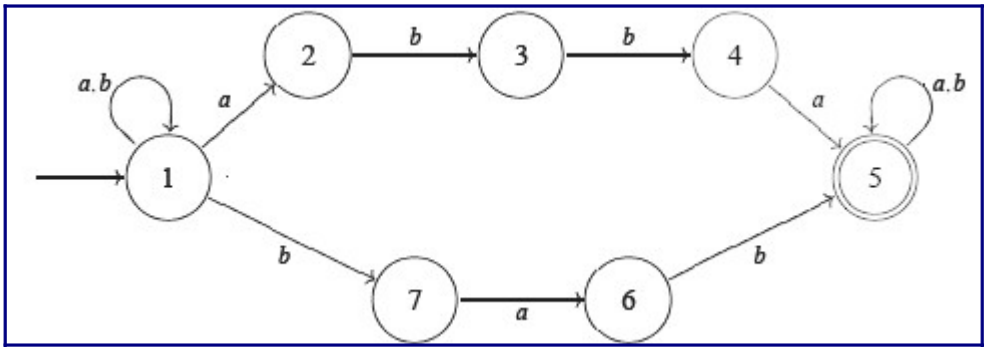
Algoritmus:

$Q' := \{ \{q_0\} \}$; $\delta' := \emptyset$; $F' := \emptyset$; $Done := \emptyset$;
while $(Q' - Done) \neq \emptyset$ **do**
$M :=$ libovolný prvek množiny $Q' - Done$
if $M \cap F \neq \emptyset$ **then** $F' := F' \cup \{M\}$ **fi**
foreach $a \in \Sigma$ **do**
$N :=$

$$\bigcup_{p \in M} \delta(p, a)$$

$Q' := Q' \cup \{N\}$
$\delta' := \delta' \cup \{((M, a), N)\}$
od
$Done := Done \cup \{M\}$
od
$M' = (Q', \Sigma, \delta', \{q_0\}, F')$

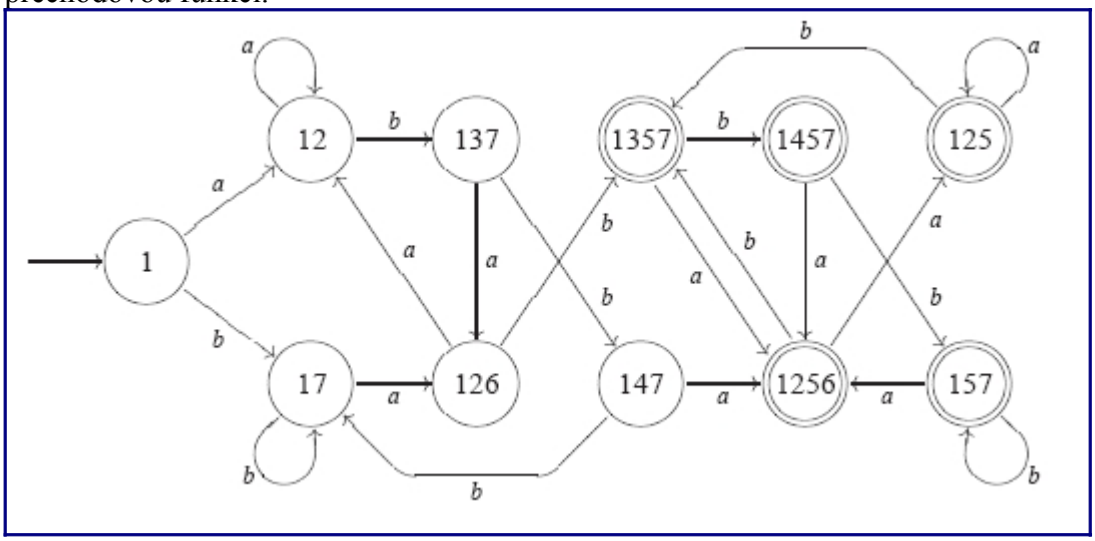
Příklad
Mějme nedeterministický konečný automat :



	a	**b**
→ **1**	1,2	1,7
2	-	3
3	-	4
4	5	-
← **5**	5	5
6	-	5
7	6	-

Začínáme vstupním stavem, který napíšeme do tabulky (**1**). Zjistíme množinu stavů, do kterých se dostaneme pomocí prvků abecedy. (**pro** *a* **1,2**, **pro** *b* **1,7**). Poté vždy vytvoříme (pokud ještě neexistuje) nový stav pojmenovaný jako sjednocení stavů (**pro** *a* **12**, **pro** *b* **17**) a zaznamenáme do tabulky. Vezmeme poté následující nezpracovaný stav v nově tvořené tabulce (**12**) a zjistíme do jaké množiny stavů se lze dostat ze stavů **1** a **2** a provedeme jejich sjednocení (**pro** *a* **1,2** , **pro** *b* **1,3,7**) poté opět zaznamenáme případný nový stav. Od této chvíle pokračujeme obdobně.
Algoritmus končí, ve chvíli kdy není možné nalézt žádný nový stav. Vstupní stavy zůstávají stejné, koncové stavy jsou ty, které obsahují některý z původních koncových stavů.

Takto vytvořený automat nemusí být minimální, ale je bez nedosažitelných stavů, s totální přechodovou funkcí.



	a	b
→ **1**	12	17
12	12	137
17	126	17
137	126	147
126	12	1357
147	1256	17
← **1357**	1256	1457
← **1256**	125	1357
← **1457**	1256	157
← **125**	125	1357
← **157**	1256	157

Podmnožinová konstrukce je na první pohled značně neefektivní – nárůst počtu stavů při přechodu od nedeterministického konečného automatu k deterministickému je exponenciální. To je z technického hlediska nepříjemné. Nabízí se proto otázka, zda použitím „pomocných" technik (odstranění nedosažitelných stavů, minimalizace) je obecně možné dosáhnout lepšího výsledku. Následující věta říká, že nikoliv.

Věta 2.44. *Pro každé $n \in \mathbb{N}$ existuje NFA o n stavech takový, že ekvivalentní DFA má i po minimalizaci 2^n stavů.*

Tedy i pro poměrně „malý“ nedeterministický konečný automat může být ekvivalentní deterministický automat „velmi velký“ i po minimalizaci. V praxi je velikost stavového prostoru samozřejmě omezená použitou technologií. Znamená to tedy, že nedeterministické konečné automaty obecně nelze efektivně implementovat? Naštěstí tomu tak není; technický „trik“, kterým lze exponenciální nárůst počtu stavů obejít, je skryt v samotné podmnožinové konstrukci: Nechť $Q = \{q_1, \ldots, q_n\}$ je množina stavů nedeterministického konečného automatu $\mathcal{M}$. Libovolnou podmnožinu $X \subseteq Q$ pak můžeme jednoznačně zakódovat bitovým řetězcem $\mathcal{B}$ délky n takto:

$$i\text{-tý prvek } \mathcal{B} = \begin{cases} 1 & \text{pokud } q_i \in X, \\ 0 & \text{jinak.} \end{cases}$$

Libovolný stav deterministického automatu $\mathcal{M}'$, který je výstupem podmnožinové konstrukce aplikované na $\mathcal{M}$, můžeme tedy reprezentovat n-bitovým řetězcem. Technická implementace automatu $\mathcal{M}'$ pak spočívá v realizaci dvou funkcí:

`function NextState(state: StateCode, symbol: Char): StateCode`

Tato funkce má dva parametry: `state` je stav automatu $\mathcal{M}'$, zakódovaný jako n-bitový řetězec (bitové řetězce jsou implementovány pomocí datového typu `StateCode`). Druhý parametr `symbol` je vstupní symbol. Funkce vrací kód stavu $\delta'(q, \texttt{symbol})$, kde q je stav s kódem `state`.

`function IsFinal(state: StateCode): Boolean`

Funkce má jeden parametr, kterým je stav automatu $\mathcal{M}'$, zakódovaný jako n-bitový řetězec. Vrací `True` pokud je stav s kódem `state` koncový, jinak `False`.

Jedinou datovou strukturou, která je potřebná pro implementaci těchto funkcí, je tabulka přechodové funkce δ původního nedeterministického automatu $\mathcal{M}$ s vyznačenými koncovými stavy.

Funkce `NextState` na základě svých parametrů a tabulky pro δ snadno určí kód výsledného stavu (i-tý bit výsledku bude nastaven na 1, právě když $q_i \in \delta(q_j, \texttt{symbol})$ pro nějaké j takové, že j-tý bit prvního parametru je nastaven na 1).

Podobně funkce `IsFinal` vrátí hodnotu `True`, pokud existuje i takové, že $q_i \in F$ a i-tý bit parametru je 1. Není tedy zapotřebí implementovat tabulku přechodové funkce δ', ani není nutné *explicitně* reprezentovat stavy automatu $\mathcal{M}'$.

Na nedeterministický automat $\mathcal{M}$ proto můžeme pohlížet jako na *symbolickou* reprezentaci automatu $\mathcal{M}'$, která dokáže popsat stavy i přechodovou funkci $\mathcal{M}'$ podstatně hutnější (úspornější) syntaxí.

Regulární jazyky

Jazyk L je **regulární**, právě když:

- může být vygenerován **regulární gramatikou** (tzn. existuje regulární gramatika G taková, že $L(G) = L$),
- je akceptovaný nějakým **deterministickým konečným automatem** (tzn. existuje deterministický konečný automat M takový, že $L(M) = L$),
- je akceptovaný nějakým **nedeterministickým konečným automatem** (tzn. existuje nedeterministický konečný automat M takový, že $L(M) = L$),
- může být popsán **regulárním výrazem** (tzn. existuje regulární výraz RE takový, že $L(RE) = L$)

Definice 2.57. Třída regulárních jazyků nad abecedou Σ, označovaná jako $R(\Sigma)$, je definována induktivně takto:

1. $\emptyset, \{\varepsilon\}$ a $\{a\}$ pro každé $a \in \Sigma$ je regulární jazyk nad Σ.
2. Jsou-li L_1, L_2 regulární jazyky nad Σ, jsou také, $L_1.L_2, L_1 \cup L_2$ a L_1^* regulární jazyky[3] nad Σ.
3. Každý regulární jazyk vznikne po konečném počtu aplikací kroků 1 a 2.

Jinými slovy $R(\Sigma)$ je nejmenší třída jazyků nad Σ splňující podmínky 1 a 2. Jazyky uvedené ad 1 se nazývají elementární, operace nad jazyky uvedené ad 2 se nazývají regulární. Je tedy vidět, že každý regulární jazyk lze popsat určením elementárních jazyků a předpisu, který určuje jak na tyto jazyky aplikovat regulární operace. Taková specifikace je cílem následující definice.

- pomocou ktorej sa definujú regulérne výrazy.

Pokud jsou A a B regulární jazyky, pak **sjednocení**, **průnik**, **rozdíl**, **zřetězení** a **iterace** jsou regulární jazyky.
Pokud je A regulární jazyk, pak **komplement** A je regulární jazyk.
Pokud je A regulární jazyk, pak **zrcadlový obraz** jazyka A je regulární jazyk.
Všetko to vyplýva z uzavretosti daných operácií nad triedou regulárnych jazykov
Všechny konečné jazyky jsou regulární.
Příklad: Iterace jazyka *{"ab", "c"}** je { *ε, "ab", "c", "abab", "abc", "cab", "cc", "ababab", ... }*

Rozhodnutelné problémy pro třídu regulárních jazyků

Mějme konečné automaty M a M'.
Následující problémy jsou rozhodnutelné:

- **ekvivalence**: jsou M a M' ekvivalentní? (platí $L(M)=L(M')$?)
- **inkluze** (jazyků): platí $L(M) \subseteq L(M')$?
- **příslušnost** (slova k jazyku): je-li dáno $w \in \Sigma^*$, platí $w \in L(M)$?
- **prázdnost** (jazyka): je $L(M) = \emptyset$?
- **univerzalita** (jazyka): je $L(M) = \Sigma^*$?
- **konečnost** (jazyka): je $L(M)$ konečný jazyk?

Regulární výrazy

Jsou výrazy, které popisují množinu slov. Jsou ekvivalentní nějakému automatu případně gramatice. Využívají právě zmíněných vlastností sjednocení, průniku, rozdílu, iterace, zřetězení a komplementu.

Jsou definovány induktivně:

1. základní regulární výrazy: ε, $\emptyset$, a pro každé a z Σ
2. E a F regulární výrazy, pak taky E.F, E+F (sjednocení) a E* jsou regulární výrazy
3. každý regulární výraz vznikne po konečném počtu aplikací kroků 1 a 2

Každý regulární výraz E nad abecedou Σ *popisuje* (jednoznačně určuje) jazyk $L(E)$ nad abecedou Σ (jazyk $L(E)$ je *sémantikou* regulárního výrazu E) podle těchto pravidel:

$$
\begin{aligned}
L(\varepsilon) &\stackrel{def}{=} \{\varepsilon\} \\
L(\emptyset) &\stackrel{def}{=} \emptyset \\
L(a) &\stackrel{def}{=} \{a\} \text{ pro každé } a \in \Sigma \\
L(E.F) &\stackrel{def}{=} L(E).L(F) \\
L(E+F) &\stackrel{def}{=} L(E) \cup L(F) \\
L(E^*) &\stackrel{def}{=} L(E)^*
\end{aligned}
$$

Na rozdíl od regulárních gramatik neobsahují regulární výrazy žádnou formu rekurze

Ekvivalenci mezi regulárními výrazy a konečnými automaty:
Pro každý jazyk generovaný nějakým regulárním výrazem existuje automat, který jej rozpoznává.
Dále platí věta, že pokud je jazyk rozpoznávaný nějakým DFA, pak je popsatelný regulárním výrazem.
Z toho dostávame **Kleeneho větu:** Libovolný jazyk je popsatelný regulárním výrazem právě když je rozpoznatelný konečným automatem.
Ekvivaletní formulace: Třída jazyků rozpoznatelných konečnými automaty je nejmenší třída, která obsahuje všechny konečné množiny a je uzavřena na sjednocení, zřetězení a iteraci.

Převody:
Regulérny výraz <-> konečný automat – pomocu regulární přechodový graf
Regulární přechodový graf <-> konečný automat. (nedeterm. s epsilon krokmi)

Lze napsat algoritmus, který rozhodne, zda jsou dva regulární výrazy ekvivalentní, tj. popisují stejný jazyk a redukovat daný jazyk na nějaký minimální automat.

Vztah mezi konečnými automaty a regulárními jazyky

Regulárny jazyk je definovaný 2x:

- pomocou regulárnej gramatiky
- pomocou konečného automatu

Z toho plyne, že **obě třídy jazyků jsou ekvivalentní**
To znamená, že k dané regulární gramatice lze sestrojit ekvivalentní (deterministický, nedeterministický, …) konečný automat a naopak.

Regulární gramatika → konečný automat

Ke každé regulární gramatice $G=(N, \Sigma, P, S)$ existuje nedeterministický konečný automat $M = (Q, \Sigma, \delta, q_0, F)$ takový, že $L(G) = L(M)$.

Myšlenka důkazu:

Stavy automatu budou odpovídat neterminálům gramatiky, tj. pro každý neterminál A bude existovat stav $\overline{A}$. Pro každé pravidlo $A \rightarrow aB$ přidáme do δ ($\overline{A}$, a) stav $\overline{B}$. Abychom se mohli vypořádat také s pravidly tvaru $C \rightarrow a$, zavedeme speciální koncový stav q_f, který přidáme do δ ($\overline{C}$, a). Počáteční stav bude $\overline{S}$, koncový stav q_f a případně také $\overline{S}$, pokud gramatika obsahuje pravidlo $S \rightarrow \epsilon$.

Konečný automat → regulární gramatika

Pro každý konečný automat $M = (Q, \Sigma, \delta, q_0, F)$ existuje regulární gramatika $G=(N, \Sigma, P, S)$ taková, že $L(M) = L(G)$.

Myšlenka důkazu:

Neterminály budou odpovídat stavům, pravidla budou simulovat přechodovou funkci. Je tu však jeden problém – pokud automat přijímá prázdné slovo (tj. počáteční stav je koncovým stavem), musí každá ekvivalentní gramatika nutně obsahovat pravidlo $S \rightarrow \epsilon$, kde S je kořen. Pak se ale S nesmí vyskytovat na pravé straně žádného pravidla. Přitom je ale možné, že některé přechody automatu končí v počátečním stavu a mají být simulovány pravidly, které mají na pravé straně S, což by vedlo ke konfliktu s požadavkem pravostranných výskytů S. Tento problém vyřešíme tak, že ke zkonstruované gramatice (mající jak $S \rightarrow \epsilon$, tak i pravostranné výskyty S) nalezneme ekvivalentní regulární gramatiku.

Aplikace regulárních jazyků a konečných automatů:

Vyhledávání vzorů v textu – editory, textové systémy (grep).
Zpracování obrazů.
Specifikace a verifikace konečně stavových automatů.