2.5 Regulární výrazy

2.5.1 Regulární jazyky jsme definovali jako ty jazyky, které jsou přijímány konečnými automaty. Ukázali jsme, že nezáleží na tom, zda jsou deterministické nebo nedeterministické. Regulární výrazy je další popis regulárních jazyků – viz 2.5.4 a 2.5.5. (Byly to právě regulární výrazy, které daly jméno třídě jazyků přijímaných konečnými automaty.)

2.5.2 Regulární výrazy nad abecedou.

Definice. Je dána abeceda Σ. Množina všech *regulárních výrazů* nad Σ je definována induktivně takto:

- $\emptyset$ je regulární výraz,
- ε je regulární výraz,
- $\mathbf{a}$ je regulární výraz pro každé písmeno $a \in \Sigma$,
- jsou-li $\mathbf{r_1}$ a $\mathbf{r_2}$ regulární výrazy, pak $(\mathbf{r_1} + \mathbf{r_2})$, $\mathbf{r_1 r_2}$ a $\mathbf{r_1^\star}$ jsou také regulární výrazy.

□

2.5.3 Jazyk reprezentující regulární výraz. Každý regulární výraz nad abecedou Σ reprezentuje jazyk nad abecedou Σ a to takto:

- Regulární výraz $\emptyset$ reprezentuje jazyk $\emptyset$.
- Regulární výraz ε reprezentuje jazyk $\{\varepsilon\}$.
- Je-li $a \in \Sigma$, pak regulární výraz $\mathbf{a}$ reprezentuje jazyk $\{a\}$.
- Jestliže regulární výraz $\mathbf{r_1}$ reprezentuje jazyk L_1 a regulární výraz $\mathbf{r_2}$ reprezentuje jazyk L_2, pak regulární výraz $(\mathbf{r_1} + \mathbf{r_2})$ reprezentuje jazyk $L_1 \cup L_2$ a regulární výraz $\mathbf{r_1 r_2}$ reprezentuje jazyk $L_1 L_2$.
- Jestliže regulární výraz $\mathbf{r}$ reprezentuje jazyk L, pak regulární výraz $\mathbf{r}^\star$ reprezentuje jazyk $L^\star$.

□

2.5.4 Věta. Každý jazyk reprezentovaný regulárním výrazem je regulární. □

Důkaz: Regulární výrazy $\emptyset$, ε, $\mathbf{a}$ (pro $a \in \Sigma$) reprezentují po řadě jazyky $\emptyset$, $\{\varepsilon\}$, $\{a\}$. Všechny tyto jazyky jsou regulární.

O třídě regulárních jazyků víme, že je uzavřena na sjednocení, zřetězení a Kleeneho operaci $\star$. To znamená, že jsou-li jazyky reprezentované regulárními výrazy $\mathbf{r}$, $\mathbf{r_1}$ a $\mathbf{r_2}$ regulární, pak jsou regulární i jazyky reprezentované regulárními výrazy $(\mathbf{r_1} + \mathbf{r_2})$, $\mathbf{r_1\, r_2}$ a $\mathbf{r}^\star$.

2.5.5 Kleeneho věta. Každý jazyk přijímaný konečným automatem je možné reprezentovat regulárním výrazem. □

Důkaz: Je dán DFA $M = (Q, \Sigma, \delta, q_0, F)$, který přijímá jazyk L. Pro jednoduchost označme množinu stavů $Q = \{1, \ldots, n\}$ a počáteční stav $q_0 = 1$. Pro $k = 0, 1, \ldots, n$ definujeme množiny slov $R_{i,j}^{(k)}$ takto:

$$R_{i,j}^{(k)} = \{w\,;\ \delta^\star(i,w) = j \text{ a sled z } i \text{ do } j \text{ má vnitřní stavy jen mezi stavy } \{1,\ldots,k\}\}.$$

Zřejmě platí:

$$R_{i,j}^{(0)} = \{a\,;\, \delta(i,a) = j\} \text{ pro } i \neq j;\ R_{i,i}^{(0)} = \{\varepsilon\} \cup \{a\,;\, \delta(i,a) = j\} \text{ pro } i = j.$$

To platí pro to, že sledy, které nemají vnitřní vrchol, jsou buď sledy o jedné hraně, nebo triviální sledy.

V obou případech se jedná o konečnou množinu. Proto umíme množinu $R_{i,j}^{(0)}$ reprezentovat regulárním výrazem.

Předpokládejme, že všechny množiny slov $R_{i,j}^{(k)}$ $(i, j \in Q)$ pro dané k umíme reprezentovat regulárním výrazem $\mathbf{r}_{i,j}^k$. Pak pro množinu slov $R_{i,j}^{(k+1)}$ platí:

$$R_{i,j}^{(k+1)} = R_{i,j}^{(k)} \cup R_{i,k+1}^{(k)} \, (R_{k+1,k+1}^{(k)})^\star \, R_{k+1,j}^{(k)}.$$

Proto množinu $R_{i,j}^{(k+1)}$ lze reprezentovat regulárním výrazem $\mathbf{r}_{i,j}^k + \mathbf{r}_{i,k+1}^k \, (\mathbf{r}_{k+1,k+1}^k)^\star \, \mathbf{r}_{k+1,j}^k$, což je opět regulární výraz.

Navíc, jazyk L je sjednocení všech množin $R_{1,j}^{(n)}$ pro $j \in F$. Proto jazyk L je reprezentován regulárním výrazem $\sum_{j \in F} \mathbf{r}_{1,j}^n$.

2.5.6 Ekvivalentní regulární výrazy. Dva různé regulární výrazy mohou reprezentovat stejný jazyk. Proto zavádímí pojem ekvivalentních regulárních jazyků.

Definice. Řekneme, že dva regulární výrazy $\mathbf{r}$ a $\mathbf{q}$ jsou *ekvivalentní*, jestliže jimi reprezentované jazyky jsou stejné. Fakt, že regulární výrazy $\mathbf{r}$ a $\mathbf{q}$ jsou ekvivalentní zapisujeme $\mathbf{r} \models\!\!\!\dashv \mathbf{q}$. □

2.5.7 Některé ekvivalentní regulární výrazy. Jsou-li $\mathbf{r}$, $\mathbf{p}$ a $\mathbf{q}$ regulární výrazy, pak

1. $\mathbf{p} + \mathbf{q} \models\!\!\!\dashv \mathbf{q} + \mathbf{p}$,
2. $(\mathbf{p} + \mathbf{q})\mathbf{r} \models\!\!\!\dashv \mathbf{pr} + \mathbf{qr}$, $\mathbf{r}(\mathbf{p} + \mathbf{q}) \models\!\!\!\dashv \mathbf{r}\,\mathbf{p} + \mathbf{r}\,\mathbf{q}$,
3. $(\mathbf{r}^\star)^\star = \mathbf{r}^\star$, $(\varepsilon + \mathbf{r})\,(\varepsilon + \mathbf{r})^\star \models\!\!\!\dashv \mathbf{r}^\star$,
4. $(\mathbf{p} + \mathbf{q})^\star \models\!\!\!\dashv (\mathbf{p}^\star\mathbf{q}^\star)^\star \models\!\!\!\dashv (\mathbf{p}^\star + \mathbf{q})^\star \models\!\!\!\dashv (\mathbf{p}^\star\mathbf{q})^\star\mathbf{p}^\star$,
5. $\mathbf{r}^\star \models\!\!\!\dashv \varepsilon + \mathbf{r}\mathbf{r}^\star$,
6. $\mathbf{r}\mathbf{r}^\star \models\!\!\!\dashv \mathbf{r}^\star\mathbf{r}$,
7. $(\mathbf{p}\,\mathbf{q})^\star \models\!\!\!\dashv \varepsilon + \mathbf{p}\,(\mathbf{q}\,\mathbf{p})^\star\,\mathbf{q}$,
8. $(\mathbf{p}\,\mathbf{q})^\star\,\mathbf{p} \models\!\!\!\dashv \mathbf{p}\,(\mathbf{q}\,\mathbf{p})^\star$,
9. $\mathbf{r}\,\emptyset \models\!\!\!\dashv \emptyset \models\!\!\!\dashv \emptyset\,\mathbf{r}$.

Poznámka. Pomocí výše uvedených vztahů není vždy jednoduché zjistit, zda dva regulární výrazy jsou ekvivalentní. Algoritmicky se daný problém řeší např. tak, že k regulárním výrazům $\mathbf{r}$ a $\mathbf{q}$ najdeme konečné deterministické automaty, které přijímají jazyky reprezentované těmito regulárními výrazy. Získané automaty následně zredukujeme. Jsou-li redukované automaty isomorfní, jsou regulární výrazy $\mathbf{r}$ a $\mathbf{q}$ ekvivalentní, jinak ekvivalentní nejsou.

2.5.8 Konstrukce konečného automatu k regulárnímu výrazu. Je dán regulární výraz $\mathbf{r}$. Sestrojit k němu konečný deterministický automat, který přijímá jazyk $L_\mathbf{r}$ reprezentovaný regulárním výrazem $\mathbf{r}$, můžeme dvěma způsoby:

I. Pomocí důkazu Kleeneho věty. Regulární výraz $\mathbf{r}$ rozdělíme na několik jednoduchých podvýrazů, pro které umíme jednoduše sestrojit NFA. Nyní pomocí metod z důkazu věty 2.4.6 sestrojíme nedeterministický automat M_1 (je-li to nutné, tak s ε přechody), který přijímá jazyk $L_\mathbf{r}$. Podmnožinovou konstrukcí k němu sestrojíme deterministický automat M_2 přijímající stejný jazyk. Je-li potřeba, M_2 redukujeme a tím dostaneme hledaný automat.

II. Přímou metodou. Jednotlivé vstupní symboly regulárního výrazu $\mathbf{r}$ očíslujeme. Dále zjistíme

1. všechny očíslované symboly, kterými může některé slovo jazyka $L_\mathbf{r}$ začínat;

2. všechny dvojice očíslovaných symbolů, které mohou po sobě následovat v některém slově jazyka $L_{\mathbf{r}}$;
3. všechny očíslované symboly, kterými může některé slovo jazyka $L_{\mathbf{r}}$ končit;
4. zda prázdné ε slovo leží v jazyce $L_{\mathbf{r}}$.

Stavy nedeterministického automatu $\overline{M}$ jsou všechny očíslované symboly $\mathbf{r}$ a počáteční stav s. Stavový diagram automatu $\overline{M}$ dostaneme takto:

Ze stavu s vede hrana do každého stavu a_i, $a \in \Sigma$, který je vyjmenován v bodě 1. Hrana je v tomto případě označena symbolem a.

Ze stavu a_i do stavu b_j $(a, b \in \Sigma)$ vede hrana označená b právě tehdy, když dvojice $a_i b_j$ byla vyjmenovaná v bodě 2.

Koncové stavy jsou všechny a_k, které jsou vyjmenované v bodě 3. Jestliže prázdné slovo ε patří do jazyka $L_{\mathbf{r}}$, je s nejen počátečním, ale i koncovým stavem.

DFA přijímající jazyk $L_{\mathbf{r}}$ dostaneme podmnožinovou konstrukcí z automatu $\overline{M}$ a případnou redukcí.

Obě metody ukážeme na příkladě.

2.5.9 Příklad. Pro regulární výraz

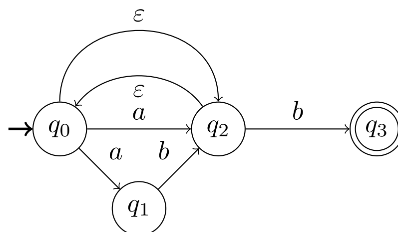
$$\mathbf{r} = (\mathbf{a} + \mathbf{ab})^\star \mathbf{b}.$$

Řešení.

I. metoda. Regulární výraz $\mathbf{r}$ rozdělíme na $\mathbf{r}_1 = (\mathbf{a} + \mathbf{ab})$, $\mathbf{r}_2 = \mathbf{b}$. Pak

$$\mathbf{r} = \mathbf{r}_1^\star \mathbf{r_2}.$$

Pak jeden ε-NFApřijímající jazyk $L_{\mathbf{r}}$ je dán stavovým diagramem



a tabulkou

	ε	a	b
$\rightarrow$ q_0	$\{q_2\}$	$\{q_1, q_2\}$	$\emptyset$
q_1	$\emptyset$	$\emptyset$	$\{q_2\}$
q_2	$\{q_0\}$	$\emptyset$	$\{q_3\}$
$\leftarrow$ q_3	$\emptyset$	$\emptyset$	$\emptyset$

Víme, že ε-UZ$(q_0) = \{q_0, q_2\} = \varepsilon$-UZ$(q_2)$, ε-UZ$(q_1) = \{q_1\}$ a ε-UZ$(q_3) = \{q_3\}$.

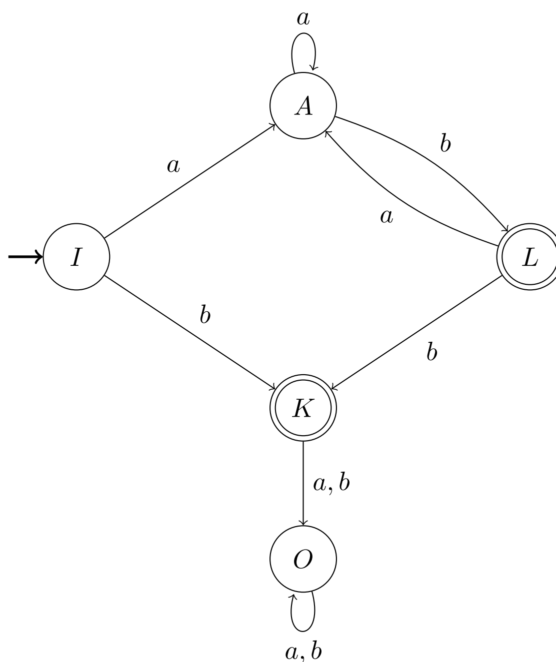
Proto podmnožinovou konstrukcí dostaneme DFA:

		a	b
$\rightarrow$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_3\}$
	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_2, q_3\}$
$\leftarrow$	$\{q_3\}$	$\emptyset$	$\emptyset$
$\leftarrow$	$\{q_0, q_2, q_3\}$	$\{q_0, q_1, q_2\}$	$\{q_3\}$
	$\emptyset$	$\emptyset$	$\emptyset$

Automat je redukovaný a po přejmenování stavů má výsledný DFA tabulku:

		a	b
$\rightarrow$	I	A	K
	A	A	L
$\leftarrow$	K	O	O
$\leftarrow$	L	A	K
	O	O	O

a stavový diagram

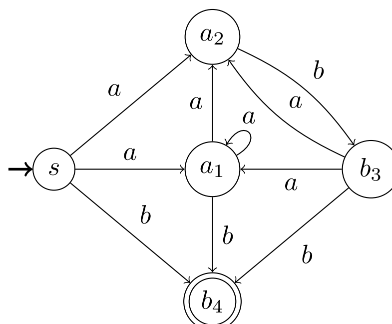


II. metoda. Očíslujeme jednotlivé výskyty vstupních symbolů:

$$(a_1 + a_2 b_3)^\star b_4.$$

1. Slovo z jazyka $L_{\mathbf{r}}$ může začínat: a_1, a_2 nebo b_4.
2. Po sobě mohou následovat (ve slově z $L_{\mathbf{r}}$): a_1a_1, a_1a_2, a_1b_4, a_2b_3, b_3a_1, b_3a_2 a b_3b_4.
3. Slovo z $L_{\mathbf{r}}$ končí b_4.
4. Prázdné slovo neleží v $L_{\mathbf{r}}$.

Zkonstruujeme nedeterministický automat podle II. metody. Ten má stavový diagram



Podmnožinovou konstrukcí dostaneme automat s tabulkou

		a	b
$\rightarrow$	s	$\{a_1, a_2\}$	$\{b_4\}$
	$\{a_1, a_2\}$	$\{a_1, a_2\}$	$\{b_3, b_4\}$
$\leftarrow$	$\{b_4\}$	$\emptyset$	$\emptyset$
$\leftarrow$	$\{b_3, b_4\}$	$\{a_1, a_2\}$	$\{b_4\}$
	$\emptyset$	$\emptyset$	$\emptyset$

Automat už je redukovaný a po přejmenování stavů dostáváme stejný automat jako metodou I.

2.5.10 Regulární výraz reprezentující jazyk $L(M)$. Je dán deterministický automat M. Regulární výraz reprezentující jazyk $L(M)$ dostaneme buď postupem z důkazu Kleeneovy věty nebo úpravami stavového diagramu. Druhý způsob popíšeme:

Ke stavovému diagramu přidáme dva stavy; a to nový počáteční stav *start* a nový koncový stav fin. Dále přidáme hrany označené prázdným slovem ε ze stavu *start* do počátečního stavu automatu a z každého koncového stavu do stavu fin. Označení hran chápeme jako regulární výrazy.

S takto vzniklým orientovaným grafem provádíme úpravy:

a) (*Odstranění paralelních hran.*) Jsou-li v grafu dvě paralelní hrany označené regulárními výrazy $\mathbf{r_1}$ a $\mathbf{r_2}$, nahradíme je jednou hranou označenou regulárním výrazem $(\mathbf{r_1} + \mathbf{r_2})$.

b) (*Odstranění smyčky.*) Je-li v grafu smyčka ve vrcholu q označená regulárním výrazem $\mathbf{r}$, pak smyčku odstraníme a označení $\mathbf{s}$ libovolné hrany začínající v q změníme na $\mathbf{r}^\star\mathbf{s}$.

c) (*Odstranění vrcholu q.*) Jestliže v q není smyčka, odstraníme jej tímto postupem: Každou dvojici hran e_1 a e_2, kde e_1 končí ve vrchollu q a e_2 začíná ve vrcholu q, tj. $KV(e_1) = q = PV(e_2)$, nahradíme hranou e s $PV(e) = PV(e_1)$ a $KV(e) = KV(e_2)$. Je-li hrana e_1 označena regulárním výrazem $\mathbf{r}$ a hrana e_2 regulárním výrazem $\mathbf{s}$, je hrana e označena regulárním výrazem $\mathbf{r}\,\mathbf{s}$.

Končíme tehdy, když graf má jen dva vrcholy, a to *start* a fin, a jedinou hranu, a to ze *start* do fin. Označení této hrany je hledaný regulární výraz.

Poznamenejme, že vytvořený regulární výraz obecně závisí na pořadí, ve kterém odstraňujeme stavy (vrcholy). Můžeme proto dostat různé regulární výrazy, ale tyto výrazy jsou ekvivalentní.

2.6 Praktické použití regulárních výrazů.

Teorie regulárních jazyků patří k úhelným kamenům informatiky. Praxe často bývá méně učesaná než teorie a s regulárními jazyky to dopadlo podobně. Proto praktickému použití regulárních jazyků, regulárních výrazů a konečných automatů věnujeme samostatnou sekci.[1]

2.6.1 Kde lze potkat regulární jazyk. Nejčastější použití regulárních jazyků není ani moc nápadné. Ve většině prakticky používaných počítačových jazyků lze rozlišit části (podřetězce), tzv. lexikální elementy, též tokeny, které lze pokládat za slova z regulárního jazyka. Např. v běžných programovacích jazycích to jsou zápisy konstant, zápisy identifikátorů, klíčová slova, relační znaménka apod. Program, který daný jazyk analyzuje (syntaktický analyzátor, parser), obvykle obsahuje proceduru, které se říká lexikální analyzátor a která funguje jako konečný automat. Lexikální analyzátor při každém svém vyvolání přečte ze vstupního souboru nejdelší část, která tvoří jeden lexikální element a volajícímu programu vrátí jeho typ a hodnotu. Zbytek syntaktického analyzátoru může díky tomu být jednodušší, neboť pracuje už jen s lexikálními elementy.

Další významné použití regulárních jazyků je pro uživatele viditelnější – je to vyhledávání výskytů slov daných regulárním výrazem.

Program grep opisuje na výstup ty řádky vstupního souboru, které obsahují podřetězec popsaný regulárním výrazem.

[1] Tuto sekci napsal Jiří Demel.

Mnohé editory dovedou při operacích „vyhledej“, popř. „vyhledej a nahraď“ hledat podle daného regulárního výrazu. Toto byla mimochodem nejstarší aplikace regulárních výrazů.

V některých programovacích jazycích (Perl, AWK, Ruby) je vyhledávání (a popř. náhrada) vzoru daného regulárním výrazem součástí jazyka. V mnoha dalších jazycích jsou k dispozici knihovny.

Vyhledávání vzorů se také využívá v antivirech a v antispamových filtrech.

2.6.2 Zápis regulárních výrazů. Formální algebraický zápis regulárních výrazů, jak byl definován v 2.5.2, je vhodný pro teoretické úvahy. Pro praktické účely se používá syntax, která zejména zjednodušuje zápis množin znaků a dovoluje flexibilně vyjádřit počty opakování.

- Regulární výraz `.` (tečka) reprezentuje libovolný znak (přesněji: jazyk tvořený všemi jednoznakovými slovy).
- Pro zápis obecnější množiny znaků se používá konstrukce s hranatými závorkami. Příklady:
  - `[abc]` reprezentuje libovolný jeden ze znaků `a`, `b`, `c`.
  - `[a-f]` reprezentuje libovolný jeden ze znaků `a` až `f`.
  - `[0-9]` reprezentuje libovolnou číslici.
  - `[^abc]` reprezentuje libovolný znak jiný než `a`, `b` a `c`.
  - `[^a-f]` reprezentuje libovolný znak jiný než `a` až `f`.
- Znak `|` („svislítko“, v UNIXu roura) značí alternativu, tedy operaci, která se v algebraickém zápisu značí $+$.
- Hvězdička (Kleeneho operace) se zapisuje v řádce, nikoli jako exponent.
- Znak `+` označuje jedno nebo více opakování předcházejícího. Tedy **r**`+` $= \mathbf{rr}*$.
- Znak `?` označuje žádný nebo jeden výskyt předchozího. Tedy **r**`?` označuje to, co by se v algebraickém zápise vyjádřilo jako $(\varepsilon + \mathbf{r})$.
- Pro obecnější počty opakování předchozího znaku nebo výrazu v závorkách se používá konstrukce se složenými závorkami. Příklady:
  - **r**`{3}` značí přesně 3 opakování,
  - **r**`{3,}` značí 3 nebo více opakování,
  - **r**`{3,5}` značí 3 až 5 opakování.
- Znak `^` na začátku výrazu značí začátek textu nebo řádky.
- Znak `$` na konci výrazu značí konec textu nebo řádky.
- Některé znaky (např. `[]^$.*`) mají v praktických regulárních výrazech speciální význam popsaný výše. Potřebujeme-li některý z těchto znaků použít ve výrazu jako obyčejný znak, je třeba před tento znak zapsat obrácené lomítko, tj. takzvaně znak escapovat. Např. pro vyhledání doménového jména `fel.cvut.cz` je třeba escapovat tečky, tedy správný výraz je `fel\.cvut\.cz`, jinak by výrazu vyhovovalo také nesprávné `felxcvutbcz`.
- Některé speciální znaky (např. `{}()+?`) se bohužel v praxi používají nejednotným způsobem. V některých implementacích je nutno tyto znaky escapovat, aby měly speciální význam, jinde je naopak nutno escapovat, aby fungovaly jako obyčejné znaky bez speciálního významu.

Praktické regulární výrazy se nejčastěji používají pro hledání podslova v prohledávaném textu. Tedy např. výraz `sen` bude nalezen v textu `nadnesený`. Chceme-li hledat `sen` samotný, musíme použít výraz `^sen$`. Výraz reprezentující prázdné slovo je `^$`.

Výše uvedený popis praktických regulárních výrazů zdaleka není úplný. Pro zevrubnější poučení doporučuji text Pavla Satrapy `http://www.nti.tul.cz/~satrapa/docs/regvyr/`, kde je i tabulka odlišností syntaxe regulárních výrazů v různých implementacích.

2.6.3 Implementace regulárních výrazů jsou zhruba tří typů.

Simulace DFA je nejrychlejší. Je-li DFA, který rozpoznává jazyk daný regulárním výrazem, již sestrojen, pak samotné prohledávání vyžaduje čas $O(t)$, kde t je délka prohledávaného textu. Konstrukce DFA k danému regulárnímu výrazu ovšem vyžaduje nějaký čas a paměť, je to tedy něco jako investice, která se ne vždy vyplatí. Pro hledání v rozsáhlých textech je simulace DFA nejrychlejší ze všech tří možností.

Stav reprezentován obsahem proměnné. Přechodová funkce může být dána ve formě dvourozměrného pole, kde přečtený znak a dosavadní stav se použijí jako indexy a prvek pole obsahuje nový stav.

Simulace NFA probíhá tak, že vždy po přečtení znaku ze vstupu vypočteme množinu všech stavů, do nichž se NFA tímto znakem může dostat. Tedy z předchozí množiny „aktivních" stavů počítáme novou množinu. Automat sice je nedeterministický, ale díky konečnému počtu stavů lze snadno paralelně sledovat všechny možné větve výpočtu. Není tedy třeba žádný bactracking.[2] Simulace NFA pracuje v čase $O(qt)$, kde q je počet stavů NFA a t je délka prohledávaného textu. Vlastní prohledávání je tedy pomalejší než v případě DFA, ale počáteční konstrukce NFA je jednodušší a rychlejší než konstrukce DFA.

Existuje varianta, kdy během simulace činnosti NFA je postupně vytvářen DFA.

Rekurzivní hledání (backtracking) nevyužívá konečné automaty, namísto toho porovnává prohledávaný text přímo s regulárním výrazem a když nenajde možnost pokračovat, vrací se a zkouší porovnat prohledávaný text s další částí regulárního výrazu. Na některých regulární výrazech pracuje backtracking značně neefektivně. Např. je-li hledán výraz `a?a?a?a?a?aaaaa` v textu `aaaaa`.

Výhodou rekurzivního hledání je rychlejší start, ale zejména možnost rozšířit množinu vyhledávaných slov za hranice regulárních jazyků pomocí tzv. zpětných referencí. Každý pár kulatých závorek, kromě toho, že seskupuje to, co je uvnitř, zároveň vymezuje část vstupního textu, která se shodovala s vnitřkem závorky. Na takto zapamatovanou část vstupního textu se pak lze odvolat zápisem *číslo*. Tedy např. ve výrazu `(.*)\1` zastupuje `\1` text, který byl reprezentován výrazem `(.*)`. Výraz `(.*)\1` tedy reprezentuje jazyk tvořený všemi zdvojenými slovy, např. `bubu` nebo `holahola`. Tento jazyk není regulární, není dokonce ani bezkontextový.

Historicky první implementace regulárních výrazů (popsána v článku z roku 1968) byla založena na paralelní simulaci NFA. Jejím autorem byl Ken Thomson, jeden z otců UNIXu. Snad proto se to v UNIXu regulárními výrazy jen hemží. Otcové UNIXu teorii regulárních jazyků a konečných automatů dobře znali. První open source implementace však byla založena na backtrackingu. Díky otevřenosti se rozšířila mimo jiné do Perlu a odtud skrze PCRE (Perl Compatible Regular Expressions) do knihoven řady dalších jazyků.

Detailní popis včetně příkladu implementace a včetně historických podrobností lze najít v článku `http://swtch.com/~rsc/regexp/regexp1.html`

[2]Pozor, v jinak velmi pěkném textu Pavla Satrapy `http://www.nti.tul.cz/~satrapa/docs/regvyr/` jsou pojmy nedeterministický, deterministický a rekurzivní použity poněkud nešťastně: slovem deterministický je označena simulace NFA, nedeterministický stroj znamená rekurzivní hledání a možnost použít DFA tam vůbec není zmíněna.