# Algoritmy na efektívnu reprezentáciu pravidiel

**Jozef Valo**

*Slovenská technická univerzita, Fakulta informatiky a informačných technológií*
*Ústav informatiky a softvérového inžinierstva*
*Ilkovičova 3, 842 16 Bratislava, Slovenská republika*
*valo.jozef at gmail.sk*

*Spracované podľa:* [1] Ingargiola, G. - The RETE Algorithm

**Abstrakt.** Aby bolo možné spracovávať veľké množstvo dát, ktoré sú reprezentované pomocou pravidiel, je potrebné používať vhodné a efektívne algoritmy. Medzi efektívne algoritmy na reprezentáciu pravidiel patrí RETE, TREAT a LEAPS. Táto práca sa prevažne zaoberá najznámejším a najpoužívanejším z týchto algoritmov - RETE avšak spomínané sú aj zvyšné dva algoritmy. Okrem samotného popisu jednotlivých algoritmov sa kladie dôraz na efektívnosť písania pravidiel, ktorá môže vo veľkej miere ovplyvniť časové a pamäťové nároky daných algoritmov.

**Abstract** (in english - bude až vo finálnej verzii)

## Obsah



## 1 Úvod

Systém pravidiel (pravidlové systémy) je spôsob ako vytvoriť zložitú rozhodovaciu logiku a ako efektívne pracovať s veľkým množstvom dát. Pravidlové systémy môžu vytvárať rozhodnutia založené na stovkách tisícov faktov a to rýchlo, efektívne a spoľahlivo. Tieto rozhodnutia vytvárajú na základe používaných algoritmov, ktoré dané pravidlá spracovávajú a reprezentujú. Medzi najznámejšie a najpoužívanejšie algoritmy patrí RETE algoritmus, ktorý je používaný v mnohých pravidlových systémoch. Preto sa budem venovať najmä tomuto algoritmu. Medzi ďalšie známe algoritmy snažiace sa o čo najefektívnejšiu reprezentáciu pravidiel bezpochyby patria TREAT a LEAPS.

## 2 RETE

RETE (z latinského slova 'rete' - sieť) je efektívny algoritmus [1],[2] na porovnávanie vzorov s faktami. Je to algoritmus so sieťovou architektúrou, na základe ktorej redukuje dobu porovnávania. RETE algoritmus navrhol a v

roku 1982 publikoval Dr. Charles Forgy. Tento algoritmus sa stal základom pre veľa známych jadier expertných systémov ako sú: CLIPS, Jess, JJBoss Rules, Soar a ďalšie.

### 2.1 Fakty, pravidlá a vzory

Algoritmus RETE pracuje s faktami, vzormi a pravidlami. Tieto výrazy budem pri vysvetlovaní RETE algoritmu dosť často používať, preto pre lepšiu predstavu toho, čo sa myslí pod týmito pojmami uvádzam nasledujúce príklady [2] jednotlivých faktov, vzorov a pravidiel.

Fakty:

```
(je-kôň strela)                ;; Strela je kôň
(je-rodič strela amadeus)      ;; Strela je rodičom amadeusa
(má-cieľ obj-7 zjednodušiť)    ;; Cieľom obj-7 je zjednodušenie
(výraz obj-7 0 * 12)           ;; Výraz obj-7 je "0 * 12"
(výraz obj-11 13 + 8)          ;; Výraz obj-11 je "13 + 8"
(vek Jano 18)                  ;; Jano má 18 rokov
```

Vzory:

```
(má-cieľ ?x zjednodušiť)
(výraz ?obj 0 ?op ?cis)
(je-rodič ?r ?d)
```

Pravidlá:

```
(P1 (má-cieľ ?x zjednodušiť)
    (výraz ?x 0 + ?y)
 ==>....)                      ;; akcie, ktoré má pravidlo vykonať
(P2 (má-cieľ ?x zjednodušiť)
    (výraz ?x 0 * ?y)
 ==>....)
(P3 (je-kôň ?x)
    (je-rodič ?x ?y)
    (je-rýchly ?Y)
 ==>....)
```

### 2.2 Sieťová štruktúra

Za pomoci kompilátora jazyka vytvára RETE sieť, v ktorej sú uložené informácie o stotožnení podmienok s faktami v báze faktov. Táto sieť sa skladá z uzlov a hrán. Obsahuje štartovací uzol a ďalej jeden uzol pre každú podmienku a konjukciu podmienok. Uzly s výstupným stupňom 0 zodpovedajú pravidlám. Výstupný stupeň uzla je počet hrán, pre ktoré je daný uzol začiatočný - t.j. počet hrán, ktoré z uzla vystupujú. Keďže uzly reprezentujúce pravidlá majú výstupný stupeň rovný nule, nachádzajú sa na dne siete. Hrany siete spájajú dva uzly - začiatočný a koncový, a reprezentujú vzťahy (väzby) premenných vyskytujúcich sa v začiatočnom uzle.

Fakt, ktorý je pridávaný do bázy faktov je reprezentovaný príznakom (token). Tento príznak sa potom umiestni do štartovacieho uzla odkiaľ sa potom šíri po sieti. Príznak môže prejsť len tou hranou, pre ktorú jeho argumenty spĺňajú vzťah s danou hranou. Ak sa v uzle, ktorý zodpovedá pravidlu, splnia všetky príznaky, tak sa pravidlo vykoná. Podobne sa príznak šíri sieťou aj pri odobraní faktu z bázy faktov, akurát s tým rozdielom, že sa odstraňujú všetky jeho kópie vyskytujúce sa v uzloch. Pokiaľ sa nejaký fakt zmení, napríklad vek Mareka z 22 na 23, tak sa najprv odstráni starý fakt (vek Marek 22) a pridá sa nový fakt (vek Marek 23).

### 2.3 Alpha a Beta pamäte

RETE sieť sa skladá z 3 typov uzlov: štartovacieho uzla, jedno-vstupových "vzorových" uzlov a dvoj-vstupových "spájacích" uzlov. Štartovací uzol má ako nasledovníkov vždy jedno-vstupové uzly.
Pre každé pravidlo sa pre každý z jeho vzorov vytvoria jedno-vstupové "alpha" uzly. Každý takýto alpha uzol má priradenú alpha pamäť, v ktorej stĺpce predstavujú premenné vzoru uzla. Napríklad, ak vzor uzla je: (je-rodič ?r ?d), tak potom v alpha pamäti budú stĺpce pomenované R a D.

Potom pre každé pravidlo Pi a jeho alpha pamäte Ai,1 Ai,2 ... Ai,n (i predstavuje číslo pravidla) sa vytvoria dvoj-vstupové uzly nazývané beta uzly: Bi,1 Bi,2 ... Bi,n kde:

Bi,1 - má ľavý vstup z Ai,1 a jeho pravý vstup je z Ai,2

Bi,j - pre j väčšie rovné 2, má ľavý vstup z Bi,j-1 a pravý vstup z Ai,j+1

Pre každý takýto beta uzol Bi,j je vytvorená relácia - beta pamäť, ktorá spája relácie (pamäte) z ľavého a pravého vstupu uzla. Stĺpce beta pamäťe obsahujú názvy premenných nachádzajúcich sa v oboch vstupných reláciách uzla. Pre predstavu ak relácia ľavého vstupu a pravého vstupu je:

```
X       Y               X       Z
=========               =============
anna    6               anna    peter
jano    12              anna    jakub
                        stano   anna
```

potom výsledná relácia beta pamäte je:

```
X       Y       Z
====================
anna    6       peter
anna    6       jakub
```

### 2.4 Príklad RETE algoritmu

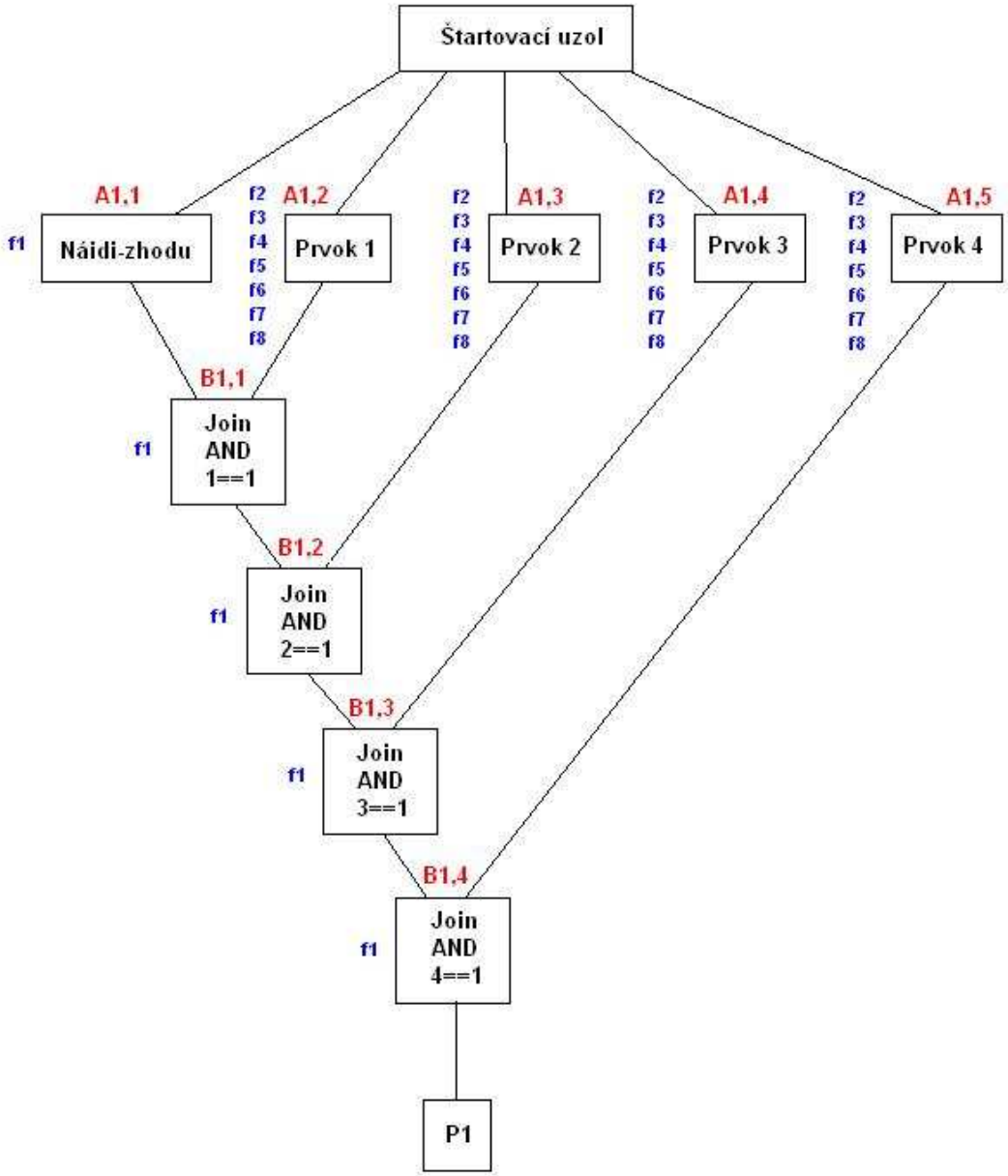
Zoberme si, že sú dané fakty:

```
FAKT                  | MENO FAKTU
==================================
(najdi-zhodu a c e g)| f1
(prvok a)             | f2
(prvok b)             | f3
(prvok c)             | f4
(prvok d)             | f5
(prvok e)             | f6
(prvok f)             | f7
(prvok g)             | f8
```

a pravidlo skladajúce sa zo vzorov:

```
PRAVIDLO 1:
    (najdi-zhodu ?x ?y ?z ?w)
    (prvok ?x)
    (prvok ?y)
    (prvok ?z)
    (prvok ?w)
    =========> ...
```

Potom jednoduchý RETE algoritmus pre dané pravidlo by vyzeral nasledovne:




Obrázok 1: RETE algoritmus.

Ako je možné vidieť na obrázku, zo štartovacieho uzla vychádza 5 výstupov, ktoré vedú do jednovstupových alpha uzlov: A1,1 - A1,5. Každý z týchto uzlov predstavuje jednu podmienku pravidla. Konkrétne napr. uzol A1,1 vyjadruje podmienku (najdi-zhodu ?x ?y ?z ?w), A1,2: (prvok ?x) atď. Ďalej sa tieto alpha uzly postupne spájajú do beta uzlov B1,1 - B1,4. Pri každom uzle sú modrou farbou vyznačené mená faktov, ktoré sú akceptované daným uzlom.

### 2.5 Efektívnosť písania pravidiel

Pri vytváraní pravidiel je dôležité sa zamyslieť nad tým ako bude toto pravidlo vyhodnocované. Pri dvoch pravidlách vykonávajúcich tú istú vec, môže pri ich rôznom zápise vzniknúť obrovský rozdiel v nárokoch na potrebnú pamäť a zároveň sa celý algoritmus značne spomalí.
Na ukázanie efektívnosti písania pravidiel nám dobre poslúži príklad uvedený v kapitole 2.4. K daným faktom a pravidlu 1 z tohto príkladu pridáme ešte pravidlo 2, ktoré má rovnaké podmienky, akurát sú zapísané v inom poradí:

```
PRAVIDLO 1:                          PRAVIDLO 2:
    (najdi-zhodu ?x ?y ?z ?w)            (prvok ?x)
    (prvok ?x)                           (prvok ?y)
    (prvok ?y)                           (prvok ?z)
    (prvok ?z)                           (prvok ?w)
    (prvok ?w)                           (najdi-zhodu ?x ?y ?z ?w)
    =========> ...                       ==========> ...
```

Ako je vidieť jediný rozdiel je v tom, že v prvom pravidle je podmienka (najdi-zhodu ?x ?y ?z ?w) na prvom mieste, zatiaľ čo v druhom pravidle je táto podmienka až na konci. Pozrime sa teraz na to, aký je medzi týmito dvomi pravidlami rozdiel. Prvé pravidlo (Obr. 1) má nasledovné nároky na pamäť:

| Uzol | Vzor | Pamäť | Počet prvkov pamäte |
|---|---|---|---|
| A1,1 | (najdi-zhodu ?x ?y ?z ?w) | f1 | 1 |
| A1,2 | (prvok ?x) | f2 f3 f4 f5 f6 f7 f8 | 7 |
| A1,3 | (prvok ?y) | f2 f3 f4 f5 f6 f7 f8 | 7 |
| A1,4 | (prvok ?z) | f2 f3 f4 f5 f6 f7 f8 | 7 |
| A1,5 | (prvok ?w) | f2 f3 f4 f5 f6 f7 f8 | 7 |
| B1,1 | a1,1 & a1,2 | f1 | 1 |
| B1,2 | b1,1 & a1,3 | f1 | 1 |
| B1,3 | b1,2 & a1,4 | f1 | 1 |
| B1,4 | b1,3 & a1,5 | f1 | 1 |
| | | spolu: | 33 |

Pre pravidlo 1 je spolu v alpha a beta pamätiach 33 prvkov. Pravidlo 2:

| Uzol | Vzor | Pamäť | Počet prvkov pamäte |
|---|---|---|---|
| A1,1 | (prvok ?x) | f2 f3 f4 f5 f6 f7 f8 | 7 |
| A1,2 | (prvok ?y) | f2 f3 f4 f5 f6 f7 f8 | 7 |
| A1,3 | (prvok ?z) | f2 f3 f4 f5 f6 f7 f8 | 7 |
| A1,4 | (prvok ?w) | f2 f3 f4 f5 f6 f7 f8 | 7 |
| A1,5 | (najdi-zhodu ?x ?y ?z ?w) | f1 | 1 |
| B1,1 | a1 & a2 | [f2 f2], [f2 f3] .. [f8 f8] | 7 * 7 = 49 |
| B1,2 | b1 & a3 | [f2 f2 f2] .. [f8 f8 f8] | 7 * 7 * 7 = 343 |
| B1,3 | b2 & a4 | [f2 f2 f2 f2] .. [f8 f8 f8 f8] | 7 * 7 * 7 * 7 = 2401 |
| B1,4 | b3 & a5 | f1 | 1 |
| | | spolu: | 2823 |

Ako je možné vidieť z daného príkladu, vzniká tu pomerne veľký rozdiel na pamäťovú náročnosť. Daný príklad obsahoval len 8 faktov, napriek tomu bol rozdiel medzi jednotlivými zápismi pravidla skoro 2800 prvkov. Aby sa pamäťové nároky čo najviac zmenšily je potrebné sa zamyslieť nad tým, ako sú dané pravidlá písané. V tomto prípade je lepšie ako prvé podmienky pravidla uviesť viac špecifické podmienky ako všeobecné. Podobne je však otázne že či je lepšie mať zložité pravidlá alebo viac jednoduchých. To už však záleží od konkrétnych prípadov, v ktorých je potrebné si zistiť podobným spôsobom ako bol uvedený pamäťové nároky.

## 3 TREAT

Treat algoritmus (Miranker 1987) [3] je algoritmus podobný RETE. Treat takisto ako RETE obsahuje štartovací a jedno a dvojvstupové uzly. Podobne ako RETE má aj alpha pamäť, akurát jednotlivé alpha pamäte sa už nespájajú do beta pamätí. Betu pamäť totiž Treat nemá. Jednotlivé alpha uzly teda neprechádzajú do beta uzlov ale ich nasledovníci sú tvorený amorfne. Znamená to, že keď príznak (token) dorazí do nejakého alpha uzla, spodná časť siete je tvorená za účelom redukcie množstva spracovaní pre dvojvstupové uzly. Pravidlo pre vytváranie siete sa nazýva "zasadací poriadok". Vždy keď príznak (token) dorazí do alpha uzla, najprv porovná príznaky v ostatných alpha uzloch. Ďalšie spracovanie sa vykonáva až v prípade, že každý alpha uzol, ktorý je potrebný pre splnenie podmienok daného pravidla, obsahuje minimálne jeden príznak. Ak niektorý alpha uzol neobsahuje žiadny príznak, je jasné že pravidlo nemôže byť splnené, tak nemá význam to ani testovať. Počet príznakov v alpha pamätiach prislúchajúcich k danému pravidlu sa kontroluje vždy pri zmene niektorej z alpha pamätí.

Tým, že Treat algoritmus nepoužíva beta pamäť má oproti RETE algoritmu oveľa menšie pamäťové nároky, čo ho predurčuje na použitie v systémoch s obmedzeným množstvom pamäte. Na druhú stranu RETE algoritmus je rýchlejší pri pridávaní jednotlivých faktov, nakoľko má už predpripravené medzivýsledky v beta pamätiach. Záleží teda od daných typov problémov / systémov ktorý z týchto algoritmov je lepšie použiť.

## 4 LEAPS

LEAPS (skr. z angl. Lazy Evaluation Algorithm for Production Systems) [4] je výkonný kompilátor súboru pravidiel pravidlového produkčného systému OPS5. Experimentálne výsledky ukazujú, že LEAPS produkuje najrýchlejšie vykonateľný súbor pravidiel OPS5. Časy vykonania môžu byť až dvakrát rádovo rýchlejšie ako u ostatných OPS5 prekladačov. Toto zrýchlenie je spôsobené v závislosti LEAPSU na komplexnej dátovej štruktúre a vyhľadávacích algoritmoch, ktoré zvyšujú pomer vykonávaných pravidiel.

### 4.1 LEAPS Kompilátor

AKo bolo skôr spomínané, LEAPS produkuje najrýchlejšie vykonateľný súbor pravidiel, často prekonávajúcich o niekoľko rádov OPS5 prekladače založené na RETE alebo TREAT algoritmoch. LEAPS prekladá OPS5 programy do C programov (Obr. 2). Ako je vidie5 z tohto obrázka je tam závislosť medzi LEAPSOM a programovacím jazykom P2. Najprv je potrebné preložiť súbor pravidiel OPS5 do P2 programu. Na to slúži RL prekladač. Tento program potom môže byť preložený do C-čka pomocou kompilátora tohto jazyka P2. Takže preklad prebieha v dvoch krokoch, čo kompilátor LEAPSU vykoná naraz. Ako je možné vidieť z tohto prekladu, všetky LEAPS algoritmy sú zahrnuté pri vytváraní P2 programu zo súboru pravidiel.

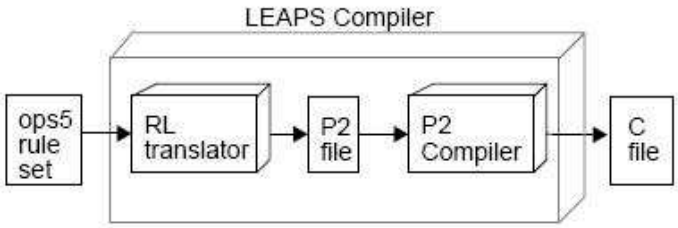



Figure 4. Relationship between LEAPS and P2

Obrázok 2: LEAPS Kompilátor. [4]

### 4.2 LEAPS algoritmus

Dopredné reťazenie, zahrnuté v LEAPSE, používa cyklus porovnanie-výber-akcia. Z pravidla, ktoré je porovnané sa vyberie 1 n-tica, ktorá korešponduje s daným pravidlom a následne sa dané pravidlo vykoná. Toto sa opakuje až kým nie je žiadne pravidlo, ktoré by sa vykonalo. RETE a TREAT algoritmy sú podstatne pomalšie z toho dôvodu, že materializujú všetky n-tice, ktoré zodpovedajú predikátom pravidla. Materializované n-tice sú uložené v dátových štruktúrach (pamätiach) a majú negatívny vplyv na výkonnosť z toho dôvodu, že musia byť obnovované po vykonaní akcií pravidla. LEAPS materializuje n-tice len vtedy, keď sú potrebné, čo redukuje časovú a pamäťovú náročnosť dopredného reťazenia.

## 5 Záver

Každé z opísaných algoritmov má svoje výhody a nevýhody. Preto je pri ich používaní sa dobre zamyslieť, aké obmedzenia má systém, v ktorom sa algoritmy používajú, a ktorý z nich bude najhodnejšie použiť. Algoritmy sa odlišujú najmä v tom aké majú pamäťové nároky, ako rýchlo dokážu spracovávať novo pridávané fakty a nad ako veľkou množinou pravidiel pracujú.

## Použitá literatúra

[1] Ingargiola, G. - The RETE Algorithm. http://www.cis.temple.edu/~ingargio/cis587/readings/rete.html#2 (13.11.2008)

[2] Forgy, C.L. 1982. Rete: a fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence, vol. 19(1): 17-37.

[3] Miranker, D. 1989. TREAT: A new and efficient match algorithm for AI production systems.

[4] Batory, D. 1994. The LEAPS Algorithm. http://www.cs.utexas.edu/ftp/pub/predator/tr-94-28.pdf (13.11.2008)

---

*Znalostné systémy*
Zimný semester 2008/2009