Porovnávanie reťazcov a syntaktická analýza

Ján Šturc

jar 2007

seminár 4.5.2007

Ciele

- Dömölkiho algoritmus je podobne ako Aho Corasickovej algoritmus zameraný na vyhľadanie mnohých vzoriek v reťazci.
- Neusiluje o asymptotické zlepšenie. Jeho zložitosť (počet porovnaní) je $O(m \times n)$, kde m je celková dĺžka vzoriek a n dĺžka reťazca. Pre porovnanie zložitosť Aho Corasickovej algoritmu je $O(n+m)$.
- Snaží sa však o to, aby porovnania prebiehali extremne rýchle. (Hoci sa to tvrdí, nie je to pravda.)
- Pre svoju jednoduchosť ja vhodný pre paralelizáciu alebo hardwareovú realizáciu.
- Robí len bitové operácie (shift, and, or a nájdenie prvej jednotky v bitovom reťazci (shift normalize).

Predspracovanie vzoriek

- K množine vzoriek priradíme booleoskú maticu, ktorá bude mať toľko riadkov, koľko znakov má abeceda (trochu menej, všetkým znakom, ktoré sa v žiadnej vzorke nevyskytujú, môžeme priradiť ten istý nulový riadok.
- Vzorky usporiadame v nejakom poradí. Usporiadaným vzorkám zodpovedajú stĺpce matice.
- M[i, j] = 1 práve vtedy, keď i-tý znak sa vyskytuje na j-tej pozícii usporiadanej vzorky.
- Vektor začiatkov jednotlivých vzoriek u[j], a vektor v[j] koncov vzoriek pre $0 \leq j \leq m$ sú definované nasledovne:
  - u[j] = 1 práve pre tie pozície j, kde nejaká vzorka začína.
  - v[j] = 1 práve pre tie pozície j, kde nejaká vzorka končí.

Algoritmus hľadania vzoriek

- Stav výpočtu je reprezentovaný vektorom q[1:m] inicializovaným ako q = **0**.
- **loop**

  c:= readinput; **if** c = EOF **then** exit loop;

  i:= ord(c); /* convert c to row index of the matrix M */

  q:= (rightshift(q) $\vee$ u) $\wedge$ M[i];

  x:= q $\wedge$ v;

  if x $\neq$ **0** **then** identify_found_patterns;

  end loop;
- rightshift je jednoduchý posun do prava s odstránením najpravejšieho bitu a doplnením nulou z ľava.
- identify_found_patterns je postupné hľadanie výskytu jedničky v x zľava. Normalizačný posun je vhodný.

Technické detaily

- Duálna podoba tohto algoritmu (jednotky a nuly a and a or sú zamenené) je známa ako Dömölki, Baeza – Yates, Gonnet SHIFT and OR algoritmus.
- Asi jediný dôvod pre duálnu podobu je inžinierská viera, že test na nulu je rýchlejší ako test na nie nulu.
- Ak sú vzorky príliš dlhé alebo strojové slovo príliš krátke možno vektory posekať na strojové slová. Treba ošetriť posun najľavejších bitov do predošlých slov ak existujú.
- Hlavným zdrojom neefektívnosti je veľmi neúsporné kodovanie stavov 2^m.

Kompakcia stavov – Aho Corasick

- Vektor q umožňuje zaznamenať súčasne všetky možné kombinácie pozície vo všetkých vzorkách aj také, ktoré nikdy nemôžu nastať. Veľká časť hodnôt vektora q je nevyužitá.
- Trie pre množinu vzoriek definuje všetky prípustné (dosiahnuteľné) stavy.
- Trie definuje aj akceptujúce stavy, stavy v ktorých nejaká vzorka bola akceptovaná a prechody pri akceptovaní vzorky. Niekedy môže byť akceptovaných viac vzoriek súčasne.
- Ostatné prechody (fail) definujeme tak, aby najdlhší suffix zlyhanej vzorky tvoril najdlhší prefix cieľovej vzorky.

Príklad – Dömölki

- Vzorky {ab, abcd, acd, dad}

matica M

	a	b	c	d	a	c	d	a	b
a	1	0	0	0	1	0	0	1	0
b	0	1	0	0	0	0	0	0	1
c	0	0	1	0	0	1	0	0	0
d	0	0	0	1	0	0	1	0	0

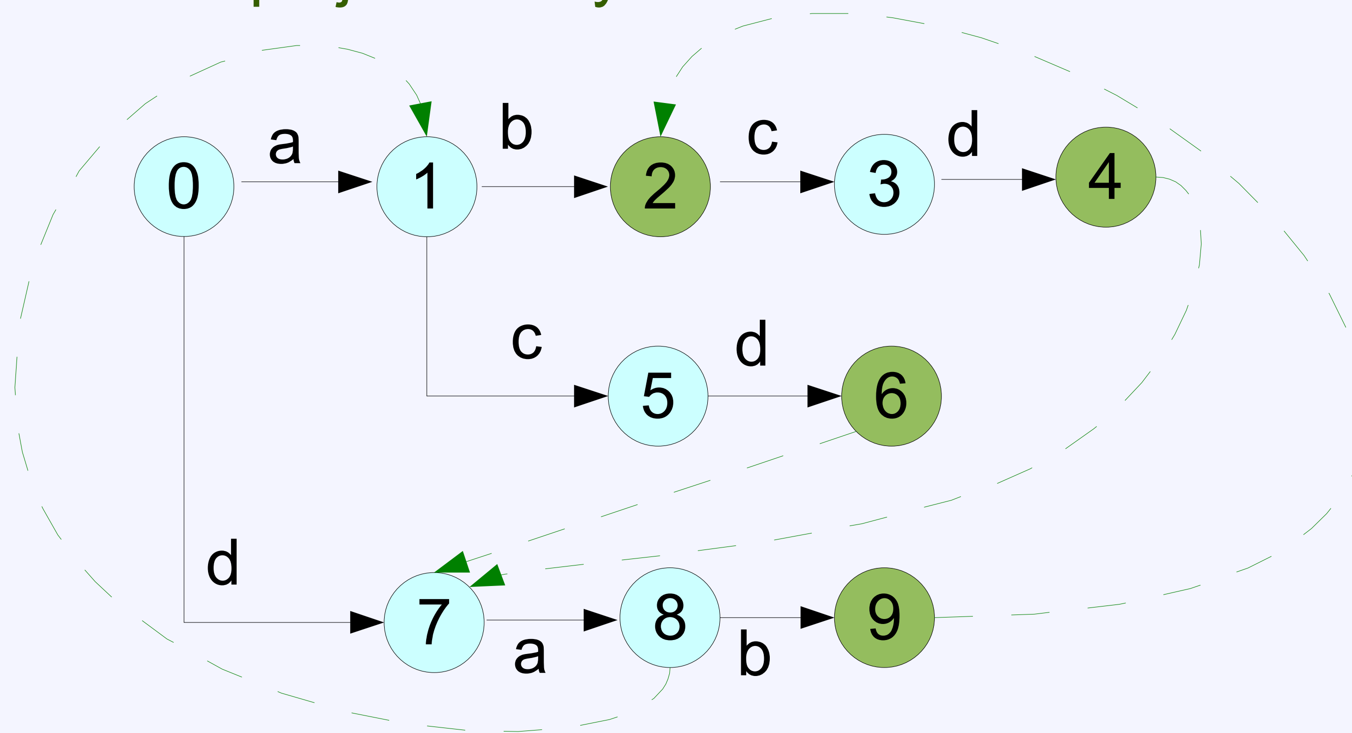
u = (1,0,0,0,1,0,1,0,0) začiatky vzoriek
v = (0,1,0,1,0,0,1,0,1) konce vzoriek

Urobil som miernú optimalizáciu, spojil som: vzorku ab a začiatok vzorky abcd a koniec vzorky acd so začiatkom vzorky dab. Takéto prelínacie optimalizácie nie sú nutné, ale môžu značne zmenšiť rozmer matice M a skrátiť dĺžku vektorov q, u, v.

Príklad – Aho Corasick

- Vzorky {ab, abcd, acd, dad}

Trie: acceptujúce stavy



Prechová tabuľka

	a	b	c	d	Fail
0	1			7	0
1		2	5		0
2			3		0
3				4	0
4					7
5				6	0
6					7
7	8				0
8		9			1
9					2

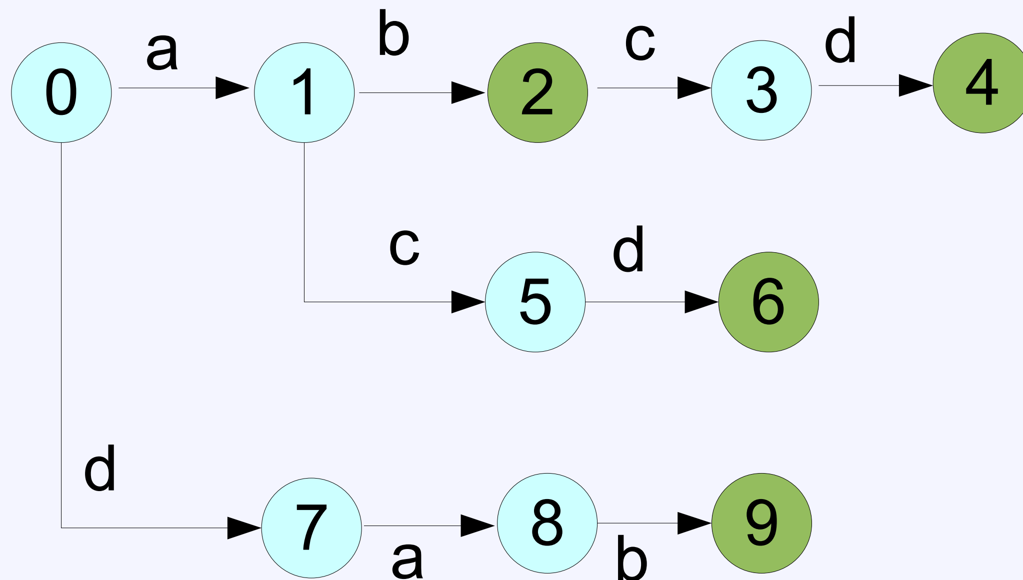
Keď chceme prechodovú tabuľku plne naplniť, môžeme ušetriť ε prechody (fail) ušetriť. Či je to výhodné, závisí od implementácie, počtu znakov abecedy a ich kódovania.

Abeceda – množiny znakov

- Obvykle (s výnimkou niektorých analýz génov) abeceda má veľa znakov a na niektorých miestách vzorky sa môže vyskytovať niekoľko rôznych znakov. Vzorky, ktoré sú takto podobné je v algoritmoch výhodné reprezentovať jednou vzorkou a nie veľa rôznými vzorkami.
- Príklad: trojciferné číslo, číslica okrem nuly, číslica, číslica
- Dömölkimu algoritmu to nespôsobuje žiadnu komplikáciu. Jednoducho v príslušnom stĺpci matice M je viac jednotiek.
- Zdanlivo ani Aho – Corasick nemá problém, označíme prechod viacerými znakmi. Pokiaľ tieto znaky sú usporiadané za sebou stačí test na dolnú a hornú hranicu. Ak to kódovanie znakov nedovolí musíme testovať každý znak vlášť. Zložitosť algoritmu sa tak zvýši na $O(k \times (n+m))$, kde k je konštanta (priemerný počet testov v jednom stave). Tabuľková realizácia tento problém nemá.
- Aho – Corasick je v podstate konečný automat a dokáže rozpoznať regulárny jazyk (cykly vo vzorkách).

Tabuľková implemntácia AC algoritmu

- Princíp: Žiaden vtip, len hrubá sila. Preč s ε-prechodmi!



Prechodová tabuľka

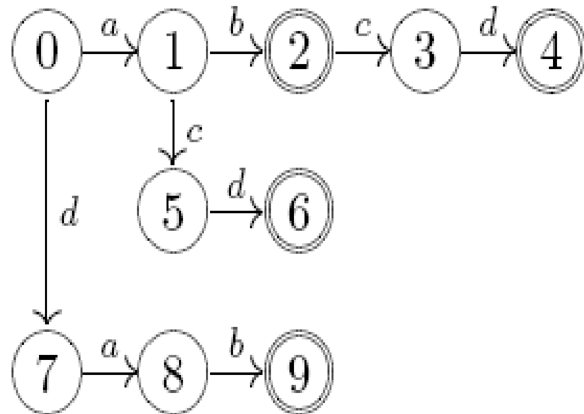
T

	a	b	c	d
0	1	0	0	7
1	1	2	5	7
2	1	0	3	7
3	1	0	0	4
4	8	0	0	7
5	1	0	0	6
6	8	0	0	7
7	8	0	0	7
8	1	9	0	7
9	1	0	3	7

Program:

```
      q:= 0;
 loop
   if q is final_state then report_patterns;
   c:=readinnput; if c=EOF then exit loop;
   q:= T[q,c];
 end loop;
```

Rôzne reprezentácie



M =

	a	b	c	d	a	c	d	a	b
a	1	0	0	0	1	0	0	1	0
b	0	1	0	0	1	0	0	1	1
c	0	0	1	0	0	1	0	0	0
d	0	0	0	1	0	0	1	0	0

$\mathbf{u} = 1\,0\,0\,0\,1\,0\,1\,0\,0$

$\mathbf{v} = 0\,1\,0\,1\,0\,0\,1\,0\,1$

T =

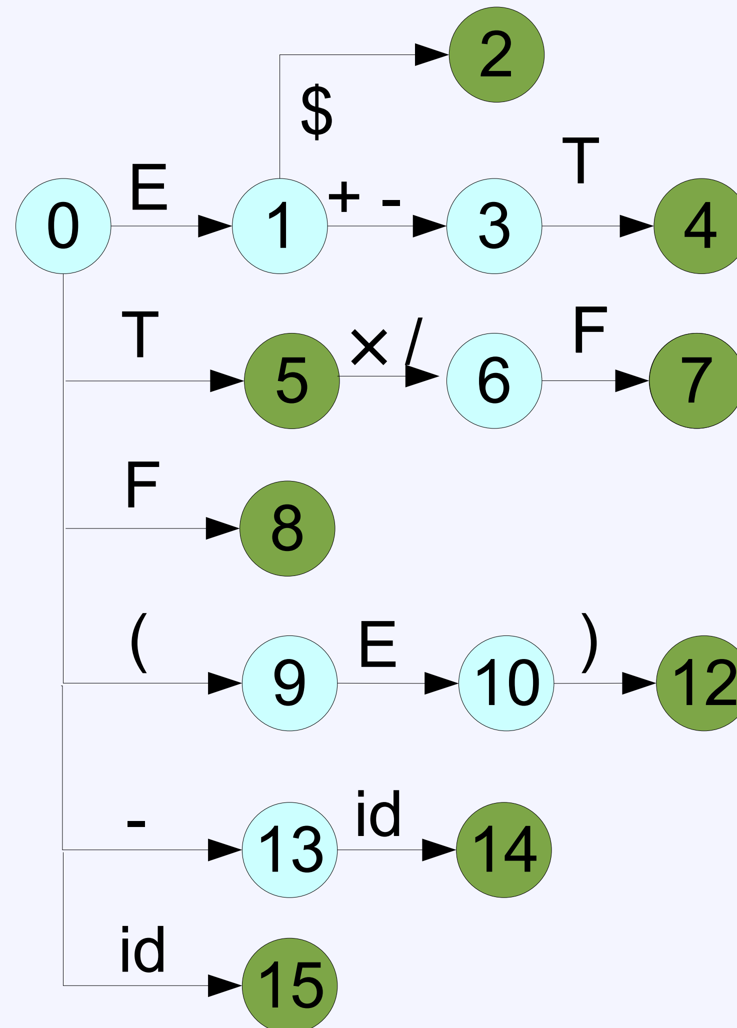
	a	b	c	d	F	pattern
0	1	0	0	7	0	
1	1	2	5	7	0	
2	1	0	3	7	1	ab
3	1	0	0	4	0	
4	8	0	0	7	1	abcd
5	1	0	0	6	0	
6	8	0	0	7	1	acd
7	8	0	0	7	0	
8	1	9	5	7	0	
9	1	0	3	7	1	ab, dab

Transformácia na algoritmus syntaktickej analýzy

- Myšlienka transformácie je založená na shift-reduce algoritmoch syntaktickej analýzy. Vzorky budú pravé strany pravidieľ gramatiky
- Pridáme zásobník a počas hľadania vzorky posúvame stavy do zásobniku (shift).
- Keď nájdeme vzorku (handle) rozhodneme sa, či máme redukovať a podľa ktorého pravidla. Pre rozhodnutie nám môže slúžiť napr. či nasledujúci symbol patrí do Follow množiny neterminálu na ľavej strane pravidla (SLR(1)).
- Ak sme sa rozhodli redukovať podľa pravidla $A \rightarrow \alpha$, vyberieme zo zásobníku $|\alpha|$ stavov (pop) a zo stavu na vrchu zásobníku „prečítame A“. (Prejdeme do stavu ako by nesledujúci vstupný symbol bol A.)
- Po redukcii pokračujeme v hľadaní na vstupe.

Príklad – aritmetický výraz

S → E$
E → E addop T | T
T → T mulop F | F
F → (E) | id | -id

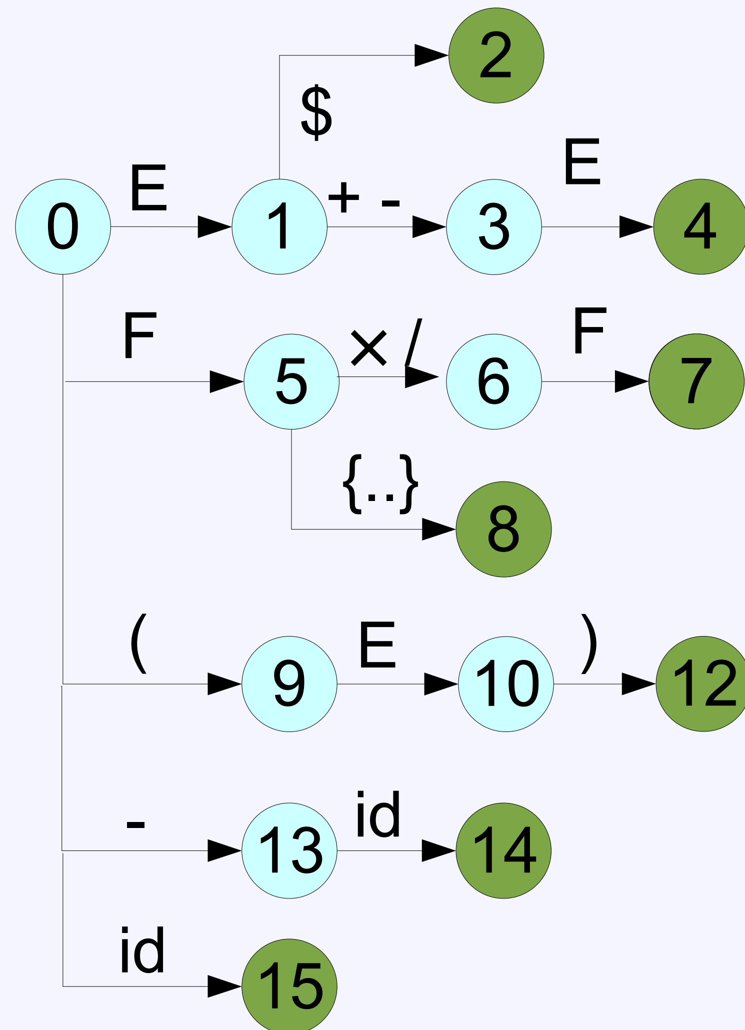


Follow množiny:
(E) {+,-,),$}
(T) {+,-,),$,×,/ }
(F) {+,-,),$,×,/ }

First(X) = {-, (, id },
pre X ∈ {S,E,T,F}.

Príklad – kontext senzitívna gramatika

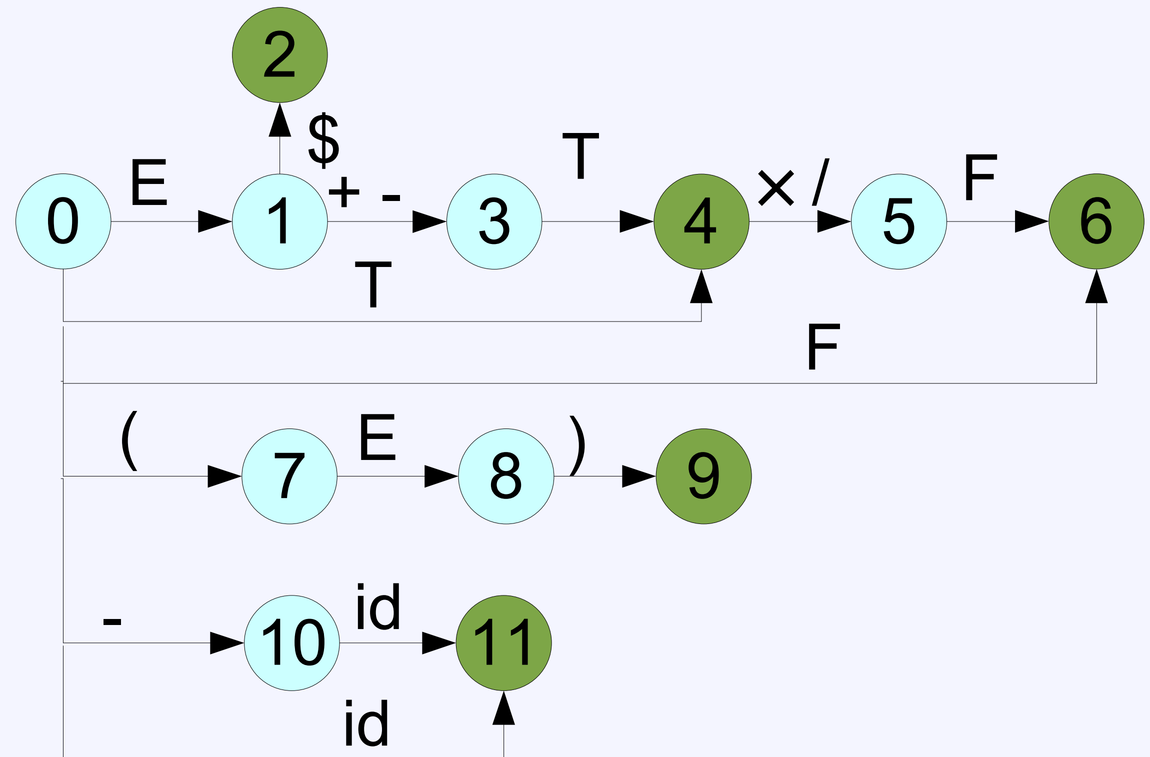
S → E$
E → E addop E
F → F mulop F
E.2 → F { addop |) | $ }
F → (E) | id | -id



aritmetický výraz, optimalizovaný

S → E$
E → E addop T | T
T → T mulop F | F
F → (E) | id | -id

Follow množiny:
(E) {+,-,),$}
(T) {+,-,),$,×,/ }
(F) {+,-,),$,×,/ }



Ak v niektorom stave je možné redukovať podľa viacerých pravidiel, rozhodujeme sa:

1. podľa predchádzajúceho stavu na zásobníku
2. v prospech dlhšieho pravidla
3. Nejak inak, ak nevieme, gramatika nie je (X)LR(1).

Príklad – Dömölkiho matica

M

	E	a	T	m	F	(	E	)	-	id	E	$
E	1	0	0	0	0	0	1	0	0	0	1	0
T	0	0	1	0	0	0	0	0	0	0	0	0
F	0	0	0	0	1	0	0	0	0	0	0	0
id	0	0	0	0	0	0	0	0	0	1	0	0
(	0	0	0	0	0	1	0	0	0	0	0	0
)	0	0	0	0	0	0	0	1	0	0	0	0
+	0	1	0	0	0	0	0	0	0	0	0	0
-	0	1	0	0	0	0	0	0	1	0	0	0
x	0	0	0	1	0	0	0	0	0	0	0	0
/	0	0	0	1	0	0	0	0	0	0	0	0
$	0	0	0	0	0	0	0	0	0	0	0	1

u = (1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0)
v = (0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0,1)

Zlučovanie susedných pozícii pri vyhľadávaní vzoriek bolo výhodné.
Tu to už nie je jasné. Ano ušetríme pri vyhľadávaní, ale platíme za to dodatočnou prácou pri určovaní podľa, ktorého pravidla sa bude redukovať.
Asi pri syntaktickej analýze táto optimalizácia nie je výhodná.
Pozor však na dĺžku strojového slova.
Pre von Neumanovský model počítača je AC lepší.

Dva zásobníky – kontextová syntaktická analýza.

- Predstavme si, že vstup je tiež zásobník a pravidlá sú tvaru: (1) $\alpha A \beta \rightarrow \alpha \omega \beta$.
- Algoritmus AC hľadá vzorku pravú stranu (zasa je to handle). Súčasne vkladáme do zásobniku symbol a stav (učebnicová syntaktická analýza).

```
s:= top(input); pop(input); q:= T(q, s);
push(stack, s); push(stack, q);
```

Ak máme redukovať podľa pravidla (1) postupujeme nasledovne:

```
for i:= |β|-1 downto 0 do { pop(stack); s:=top(stack); pop(stack);
                            push(input, s) }
for i:= |ω|-1 downto 0 do { pop(stack); pop(stack) }
q:= top(stack); push(input A);
```

Okamžité opísania

- Pred redukciou

$s_1q_1 \ldots s_iq_ia_1q_{i+1} \ldots a_kq_{i+k}w_1q_{i+k+1} \ldots w_mq_{i+k+m}b_1q_{i+k+m+1} \ldots b_nq_{i+k+m+1}$

$\uparrow c_j \ldots c_{j+z}$

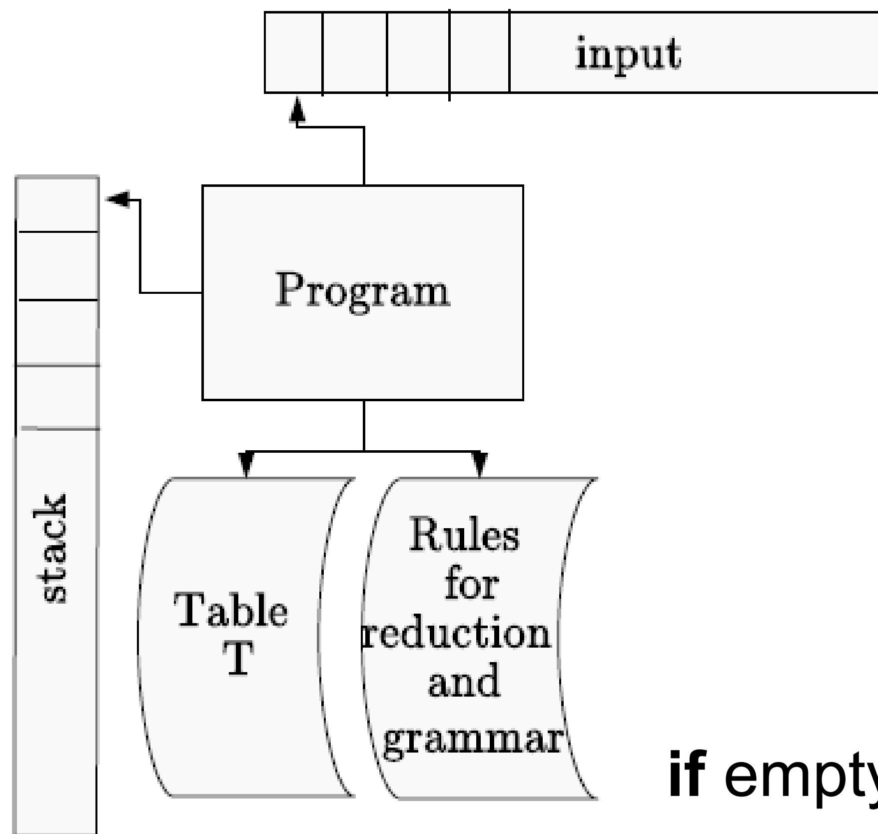
- Po redukcii

$s_1q_1 \ldots s_iq_ia_1q_{i+1} \ldots a_kq_{i+k} \uparrow Ab_1 \ldots b_nc_j \ldots c_{j+z}$

Vstupný zásobník s výnimkou ε pravidiel nerastie môžeme ho preto implementovať vracaním do vstupu (input buffer). Redukcia v tomto prípade je tak jednoduchá, že budeme túto schému používať aj pre bezkontextové gramatiky.

V tom prípade sa na vstup presúva jeden neterminál a pravý kontext.

Dvojzásobníkový automat



Program:
q:= 0; push(stack, S);
push(stack, q);
repeat
 if q ∈ F **then** Reduce;
~~else~~ **{** c := pop(input);
 push(stack, c);
 q:= T[q, c];
 push(stack, q)
 }
until q < 1;
if empty(stack) ∧ empty(input) **then**
 accept **else** error;

Procedúra Reduce

- Redukovanie môže byť podľa typu gramatiky rôzne uvedieme všeobecný prípad pravidlo $\alpha\xi\beta \rightarrow \alpha\omega\beta$, kde α, β, ξ a ω sú z $(N \cup T)^*$.

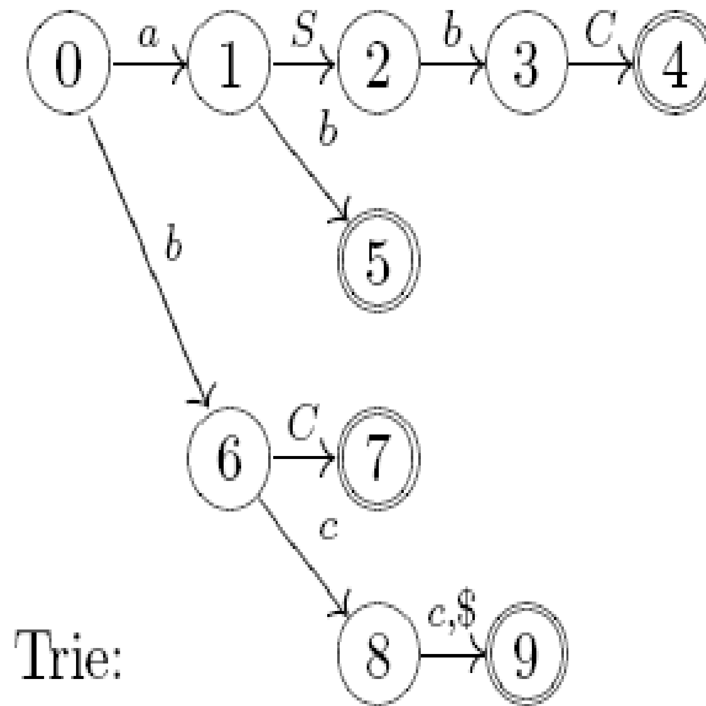
```
procedure Reduce;
 { for i := |β| downto 1 do
     { pop(stack); c:=pop(stack); push(input,c); }
   for i := |ξ| downto 1 do push(input,ξi);
   for i := 2×|ω| downto 1 do pop(stack);
   q:= top(stack);
 }
```

- Z tohto všeobecného vzoru sme vynechali jedine detail, keď niektoré symboly ξ_i sú symboly zo zásobníku. V prípade množinových pravidiel je výhodné mať možnosť adresovať tieto symboly zo zásobníku.

Príklad $\{a^n b^n c^n : n \geq 1\}$

Grammar:

$$S \rightarrow aSbC$$
$$aSb \rightarrow ab$$
$$Cb \rightarrow bC$$
$$bCc \rightarrow bcc$$
$$bC\$ \rightarrow bc\$$$



	a	b	c	$\$$	S	C	F	pattern
0	1	6					0	
1	1	5			2		0	
2	1	3					0	
3	1	6				4	0	
4							1	$aSbC$
5							1	ab
6	1	6				7	0	
7							1	bC
8	1	6	9	9			0	
9							1	$bcc, bc\$$

Vzťah k LR analýze

- Stavy pozície v trie zodpovedajú pozíciam bodky v „kernel itemoch“ LR analýzy. Prechody po trie hranách zodpovedajú „goto“ prechodom v LR analýze.
- V dôsledku toho, v prípade úspešného rozpoznania náš automat a LR (SLR(1) alebo LALR(1)) analyzátor prejde tými istými stavmi.
- LR analyzator zistí chybu, akonáhle prefix analyzovaného textu nie je životaschopný (viable) prefix žiadneho slova jazyka.
- Automat odvodený z hľadania podreťazcov zistí chybu až keď suffix okamžitého opísania nie je prefix reťazca odvoditeľného z niektorého neterminálu gramatiky.

Príklady

- aritmetický výraz: id (id+id)$
- LR: E ↑(id+id)$
- SM: FE ↑$
- $a^n b^n c^n$ výraz: aaababcbccc
- LR: aaaSb ↑abcbccc$
- SM: aaaSbS ↑Cbcc$
- Čo je lepšie vo všeobecnosti, mi nie je jasné;

Prístup LR (kompilátoru): Zistiť chybu najskôr ako sa dá.

Prístup SM (hladača vzoriek): Nájsť najviac tak dlhých ako sa dá odvoditeľných vzoriek.

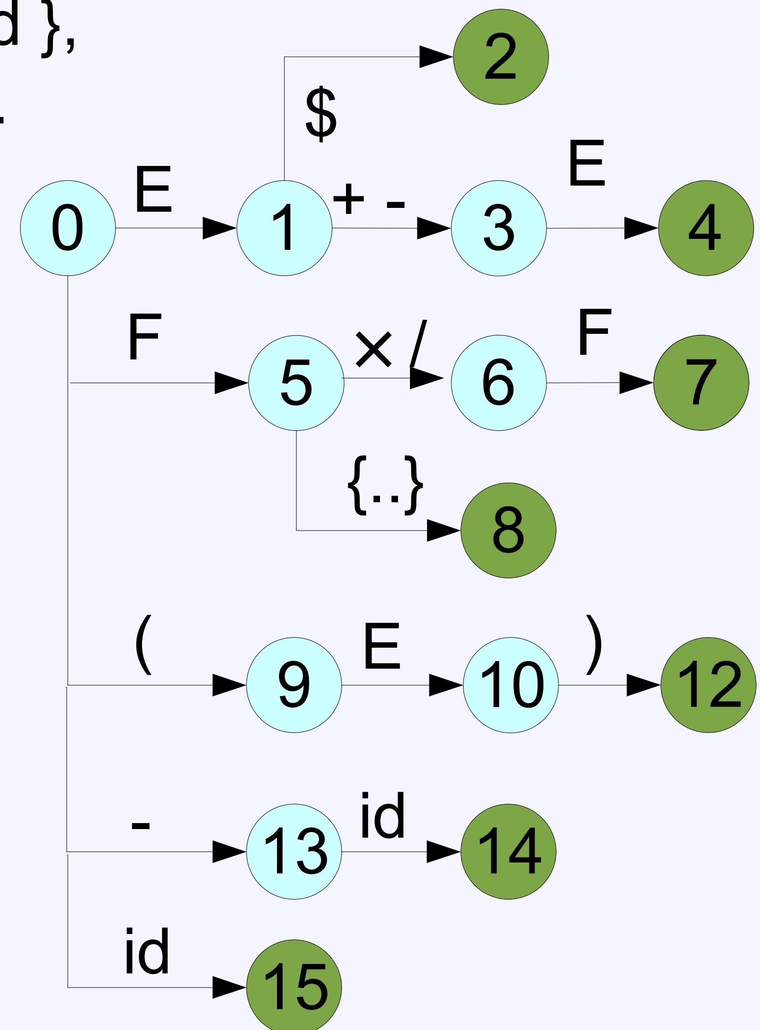
Diagnostika chýb

- Prechody po trie hranách nemôžu zodpovedať chybám.
- Ostatné prechody môžu byť chyby.
- Nie trie hrany zo stavu q_i do stavu q_j zodpovedajú korektnému odvodeniu: Ak zo stavu q_i vedie hrana označená neterminálnym symbolom X a cesta zo stavu 0 do stavu q_j je označená reťazcom r takým, že $X \overset{*}{\rightarrow} r\alpha$.
- Ostatné nie trie hrany sú chybové.
- Chybovým hranám priradíme prechody do chybových stavov (stavov označených zápornými číslami). Najlepšie, každému stavu q_j priradíme chybový stav q_{-j}.
- Predpíšeme správy a akcie pre chybové stavy. Špecialne pre q_{-j} to môže byť: Oznám chybu a pokračuj v q_j.
- Iné akcie úprava vstupu, zásobníku panický mód sú tiež možné,

Príklad – kontext senzitívna gramatika

S → E$
E → E addop E
F → F mulop F
E.2 → F { addop |) | $ }
F → (E) | id | -id

First(X) = {-, (, id },
pre X ∈ {S,E,F}.



Ostatné nechybové prechody:

	(	id	-	F
3	9	15	13	5
6	9	15	13	
9	9	15	13	5
13	9	14	?	

Diskutabílne:
T[13, '-'] ?

Pridali sme pravidlo:
F → -(E)

Chybové prechody:
1 nemali by vzniknúť
3 {+,mulop} operátor navyše
5 {id, (} chýba operátor ?

Ostatné chybové prechody

6	{+, mulop}	dva operátory za sebou
9	{+, mulop}	operátor za zátvorkou
10	{všetko okrem) }	chýba pravá zátvorka

Na rozdiel od klasických techník (automatov) syntaktickej analýzy automat založený na hľadaní vzoriek poskytuje bohaté možnosti zotavenia z chýb, ktoré organicky zapadajú do systému:

- Pre daný stav definujeme chybový stav (stavy).
- Definujeme symboly, na ktoré bude prechod do chybového stavu.
- Ak je chybový stav finálny definujeme akcie na vrcholoch pracovného a vstupného zásobníku.
- Ak to nestačí dajú sa definovať aj prechody medzi chybovými stavmi a rozpoznávať chyby.