Bottom-Up Syntax Analysis staršie - metódy

Ján Šturc

Zima 2010

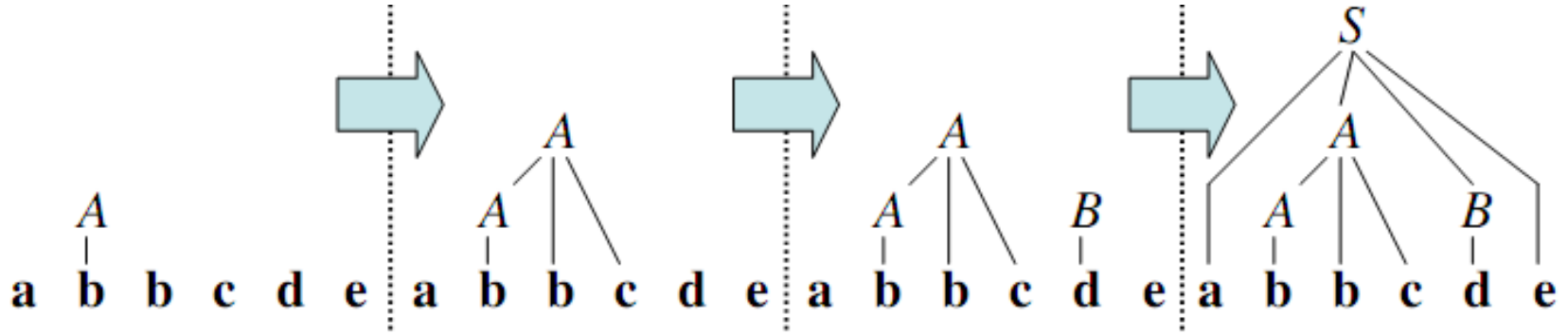
Posunovo redukčná schéma
Shift-Reduce Parsing

Grammar:
1. S → **a** A B **e**
2. A → A b **c**
3. A → **b**
4. B → **d**

Reduced sentence:
a **b** **b c d e**
a A **b c d e**
a A **d e**
a A B **e**
S

Shift-reduce corresponds to a rightmost derivation:
S $\Rightarrow_{rm}$ **a** A B **e**
$\Rightarrow_{rm}$ **a** A **d e**
$\Rightarrow_{rm}$ **a** A **b c d e**
$\Rightarrow_{rm}$ **a b b c d e**

These production's match right-hand sides: 3 2 4 1
Shift-reduce parsing produces rightmost derivation in the reverse order.



Handles

A **handle** is a substring of grammar symbols in *a right-sentential form* that matches a *right-hand side of a production*

Grammar:

1. S → **a** A B **e**
2. A → A b **c**
3. A → **b**
4. B → **d**

Reducing a sentence:

a **b** **b c d e**
a A **b c** **d e**
a A **d** **e**
a A B **e**
S

Handles:

b
A**bc**
d
aAB**e**

In the righ-sentential form **a** A **b** **c d e** the string **b** is not a handle because the result **a** A A **d e** is not a sentential form (cannot be derived from starting symbol S by the given grammar). Further reductions would fail.

Posunovo redukčný automat

- Je to zásobníkový automat, ktorý na základe vstupého symbolu a stavu, v ktorom sa nachádza vykoná jednu z akcií
  1. Shift: posunie symbol zo vstupu na zásobník a posunie vstup
  2. Reduce: redukuje symboly na vrchole zásobníku
  3. Accept: akceptuje
  4. Ak nemôže vykonať ani jednu z uvedených akcií, vykoná spracovanie chyby: Error.

Činnosť posunovo redukčného automatu

Grammar:

E → E + E

E → E * E

E → (E)

E → id

Stack	Input	Action
$	id+id*id$	shift
$id	+id*id$	reduce $E \rightarrow$ id
$E	+id*id$	shift
$E+	id*id$	shift
$E+id	*id$	reduce $E \rightarrow$ id
$E+E	*id$	shift (or reduce?)
$E+E*	id$	shift
$E+E*id	$	reduce $E \rightarrow$ id
$E+E*E	$	reduce $E \rightarrow E * E$
$E+E	$	reduce $E \rightarrow E + E$
$E	$	accept

Handles to reduce

A conflict. How to resolve it?

Konflikty – Conflicts

- Shift-reduce and reduce-reduce conflicts are caused by
  - The limitations of the parsing method (even when the grammar is unambiguous)
  - Ambiguity of the grammar

Operátorovo precedenčné gramatiky

- Neobsahujú ε- ové pravidlá
- Žiadna pravá strana neobsahuje dva po sebe idúce neterminály.
- Ideálne striedajú sa terminálne a neterminálne symboly
- Neterminálne symboly nie sú dôležité. Jeden neterminál stačí.

Príklad: aritmetický výraz

E → E + E | E - E | E*E | E/E | E↑E | (E) | id

Precedenčné relácie

- Definujeme tri rôzne precedenčné relácie:
  - **a** $\gtrdot$ **b** a „má väčšiu precedenciu ako“ b
  - **a** $\doteq$ **b** a „má rovnakú precedenciu ako“ b
  - **a** $\lessdot$ **b** a „má menšiu precedenciu ako“ b
- Relácia medzi dvojicou terminálov sa určuje:
  - na základe matematickej intuície o priorite a asociatívnosti operátorov
  - Zostrojením jednoznačnej gramatiky a použitím mechanickej metódy na odvodenie precedenčných relácií
- Precedenčné relácie slúžia aj na ohraničenie handle
  - $\lessdot$ začiatok handle
  - $\gtrdot$ koniec handle
  - Ak a $\doteq$ b, potom sa symboly a a b vyskytujú v tej istej handle

Precedenčná posunovo redukčná schéma

- Konfigurácia posunovo redukčného automatu pre precedenčnú gramatiku je tvaru:

  $\$\beta_0 a_1 \beta_1 \ldots a_m \beta_m, a_{m+1} a_{m+2} \ldots a_n \$$, kde β_i je ε alebo neterminál a a_i je terminál.
- Ak $a_m \gtrdot a_{m+1}$ (posledný operátor na zásobniku má väčšiu prioritu ako operátor na vstupe) redukujeme.
- Pri redukcii vyberáme symboly zo zásobniku (pop) pokiaľ neplatí $a_k \lessdot a_{k+1}$ (operátor zostávajúci na zásobníku má menšiu prioritu ako posledný vybratý operátor.
- Neterminály neovplyvňujú parsovanie. Pre popis gramatiky stačí jeden a pri parsovaní ich nepotrebujeme vôbec.

Algoritmus precedenčnej analýzy

```
set ip na začiatok vstupu w$;
loop
    if top=$ and ip↑ = $ then EXIT
          else  {   a := top; b := ip ↑;
                    if  a ⋖ b or a ≐ b  then push; advance(ip)
                         elseif a ⋗ b then  /*reduce*/
                               { repeat pop
                                  until top ⋖ symbol last popped;
                                  output(rule)
                                }
                         else error
                    }
pool
```

Intuitívna konštrukcia precedenčných relácii

1. O_1 má väčšiu prioritu ako O_2
   - $O_1 \dot{>} O_2$
   - $O_2 \dot{<} O_1$
     - Príklad: + < *, * >+
2. O_1 a O_2 majú rovnakú prioritu (alebo O_1 a O_2 je ten istý operátor)
   - $O_1 \dot{>} O_2$, $O_2 \dot{>} O_1$ ak operátory sú ľavo asociatívne
   - $O_1 \dot{<} O_2$, $O_2 \dot{<} O_1$ ak operátory sú pravo asociatívne
     - Príklad: $+ \dot{>} +$, $+ \dot{>} -$, $- \dot{>} +$, $- \dot{>} -$, $\uparrow \dot{<} \uparrow$

Operátory a ostatné tokeny

- Nech O je ľubovolný operátor. Potom
  - $O \lessdot$ **id**, **id** $\gtrdot O$, $\$ \lessdot O$ $O \gtrdot \$$,
  - $O \lessdot ($, $(\lessdot O$
  - $O \gtrdot)$, $) \gtrdot O$
- Relácie neobsahujúce operátory
  - $(\doteq)$ $\$ \lessdot (\$ \lessdot \mathrm{id}$
  - $(\lessdot ($ $\mathrm{id} \gtrdot \$$ $) \gtrdot \$$
  - $(\lessdot \mathrm{id}$ $\mathrm{id} \gtrdot)$ $) \gtrdot)$

Precedenčná tabuľka – príklad

Gramatika:

E → E + E | E - E | E*E | E/E | E↑E | (E) | **id**

	+	-	*	/	↑	id	(	)	$
+	· >	· >	< ·	< ·	< ·	< ·	< ·	· >	· >
-	· >	· >	< ·	< ·	< ·	< ·	< ·	· >	· >
*	· >	· >	· >	· >	< ·	< ·	< ·	· >	· >
/	· >	· >	· >	· >	< ·	< ·	< ·	· >	· >
↑	· >	· >	· >	· >	< ·	< ·	< ·	· >	· >
id	· >	· >	· >	· >	· >			· >	· >
(	< ·	< ·	< ·	< ·	< ·	< ·	< ·	≐	
)	· >	· >	· >	· >	· >			· >	· >
$	< ·	< ·	< ·	< ·	< ·	< ·	< ·		

Kompresia precedenčnej tabuľky

dvojica funkcií f, g.

f(a) < g(b), ak a $\lessdot$ b

f(a) = g(b), ak a $\doteq$ b

f(a) > g(b), ak a $\gtrdot$ b

Príklad:

	+	-	*	/	↑	id	(	)	$
f	2	2	4	4	4	0	6	6	0
g	1	1	3	3	5	5	0	5	0

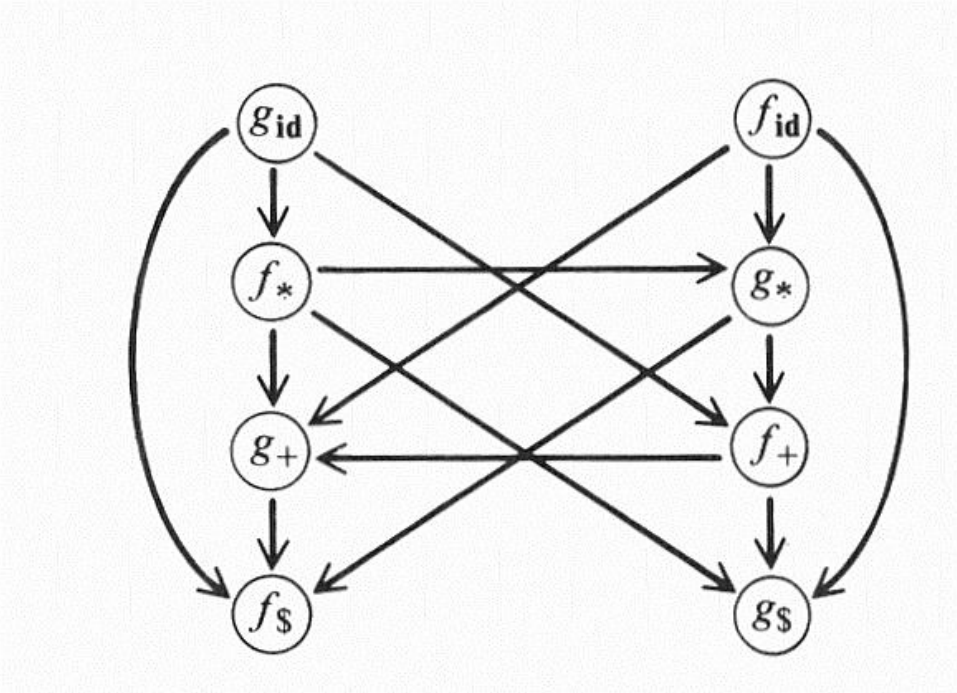
Konštrukcia funkcií f a g

1. Vytvor symboly fa a ga pre všetky a ∈ T ∪ { $ }
2. Zlučuj skupiny: Na počiatku je každý symbol jedna skupina.
   - Ak a ≐ b zlúč skupinu fa a gb.
3. Zo skupín urob uzly orientovaného grafu.
4. Pridaj hrany
   - Ak a ⋖ b ⇒ hrana (gb) → (fa)
   - Ak a ⋗ b ⇒ hrana (fa) → (gb).
5. Ak graf je cyklický precedenčné funkcie neexistujú.
6. Ak graf je acyklický

   f(a)
g(a) } = dĺžka najdlhšej cesty vychádzajúcej z fa resp. ga

Príklad

	+	*	id	$
+	⋗	⋖	⋖	⋗
*	⋗	⋗	⋖	⋗
id	⋗	⋗	⋗	⋗
$	⋖	⋖	⋖	
f	2	4	4	0
g	1	3	5	0



Syntaktická analýza zdola nahor a vyhľadávanie reťazcov

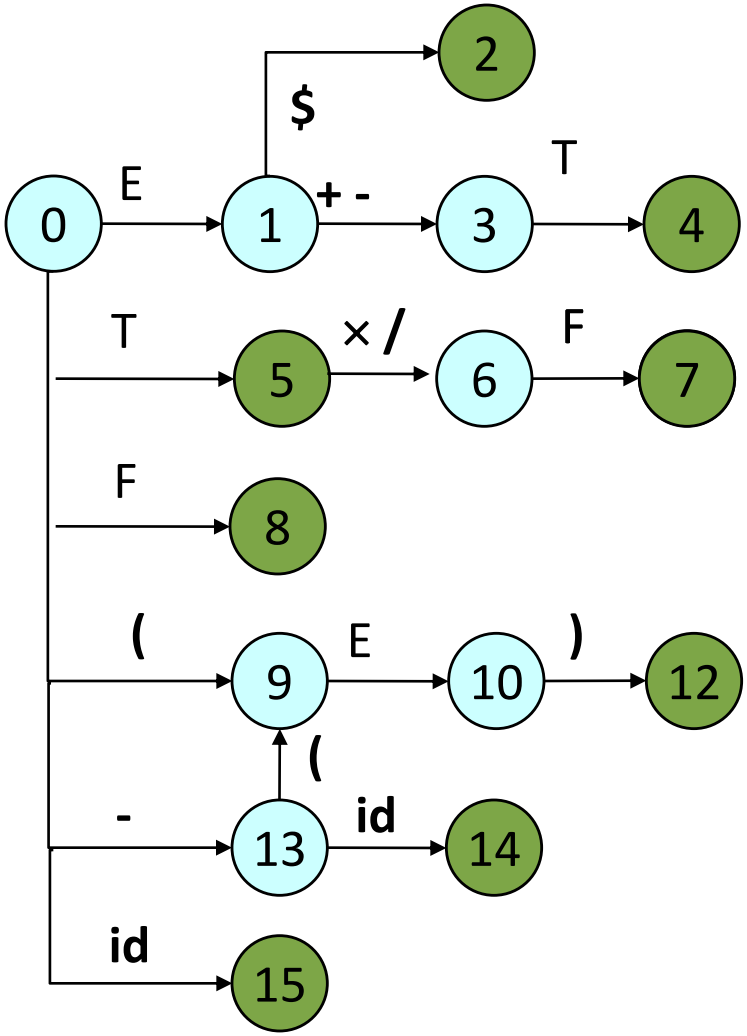
- Bottom-Up parsing hľadá výskyt pravej strany pravidla vo vetnej forme (viable prefix).
- Idea je použiť štandardný algoritmus na paralné vyhľadávanie vzoriek (Aho-Corasick, Dömölki)
- V príslušných automatoch stačí doplniť prácu so zásobníkom
- Keďže sa nechceme obmedziť na bezkontextové jazyky zavedieme zásobníky dva
  - vstupný (na vracanie už čiastočne spracovaného textu na vstup)
  - pracovný (normálny zásobník posunovo redukčného automatu)

Príklad: Aho Corasickovej automat pre gramatiku aritmtického výrazu

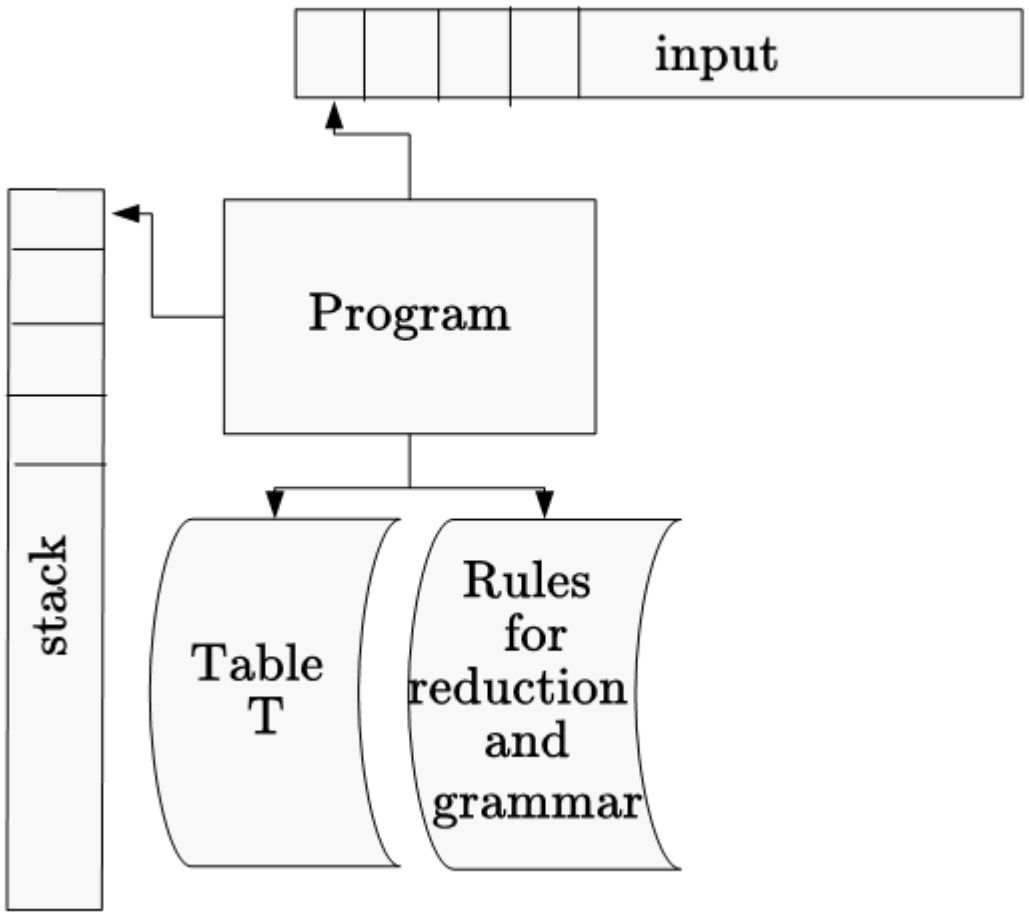
S → E$
E → E addop T | T
T → T mulop F | F
F → (E) | -(E) | id | -id

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

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



Dvojzásobníkový automat



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

Analýza kontext senzitívnych jazykov

- Pravidlá sú tvaru: (1) $\alpha A \beta \rightarrow \alpha \omega \beta$.
- Algoritmus AC hľadá vzorku pravú stranu ($\alpha\omega\beta$ je to handle).
- Rovnako ako pri LR-like vkladáme do zásobniku symbol a stav.
- Shift: s:= top(input); pop(input);
  q:= T[q, s];
  push(stack, s); push(stack, q);
- Reduce (podľa pravidla 1):
  for i:= |β|-1 **downto** 0 **do {** pop(stack); s:=top(stack); pop(stack);
  push(input, s) **}** /* vráti na vstup β */
  for i:= |ω|-1 **downto 0 do {** pop(stack); pop(stack) **}**
  / * vypopuje z pracovného zásobníku ω */
  q:= top(stack); push(input A); /* Dá na vstup A */

Konfigurácie

- Konfigurácie sú podobné konfiguráciam posunovo redukčného automatu uvedieme ich pre prípad redukcie:
- Pred redukciou:

$$s_1q_1 \dots s_iq_ia_1q_{i+1} \dots a_kq_{i+k}w_1q_{i+k+1} \dots w_mq_{i+k+m}b_1q_{i+k+m+1} \dots b_nq_{i+k+m+1}$$
$$\uparrow c_j \dots c_{j+z}$$

- Po redukcii

  $s_1q_1 \dots s_iq_ia_1q_{i+1} \dots a_kq_{i+k} \uparrow Ab_1 \dots .b_nc_j \dots c_{j+z}$
- Pre lineárne ohraničené jazyky vstupný zásobník nikdy nebude väčší ako pôvodný vstup

Špeciálny prípad β = ε

- Pravidlá tvaru (2) $\alpha A \rightarrow \alpha\omega$.
- Posun funguje rovnako ako v predošlom prípade nič sa nemení.
- Pri redukcii je prvý cyklus prázdny zostane:

  for i:= |ω|-1 **downto 0 do {** pop(stack); pop(stack) **}**

  / * vypopuje z pracovného zásobníku ω */

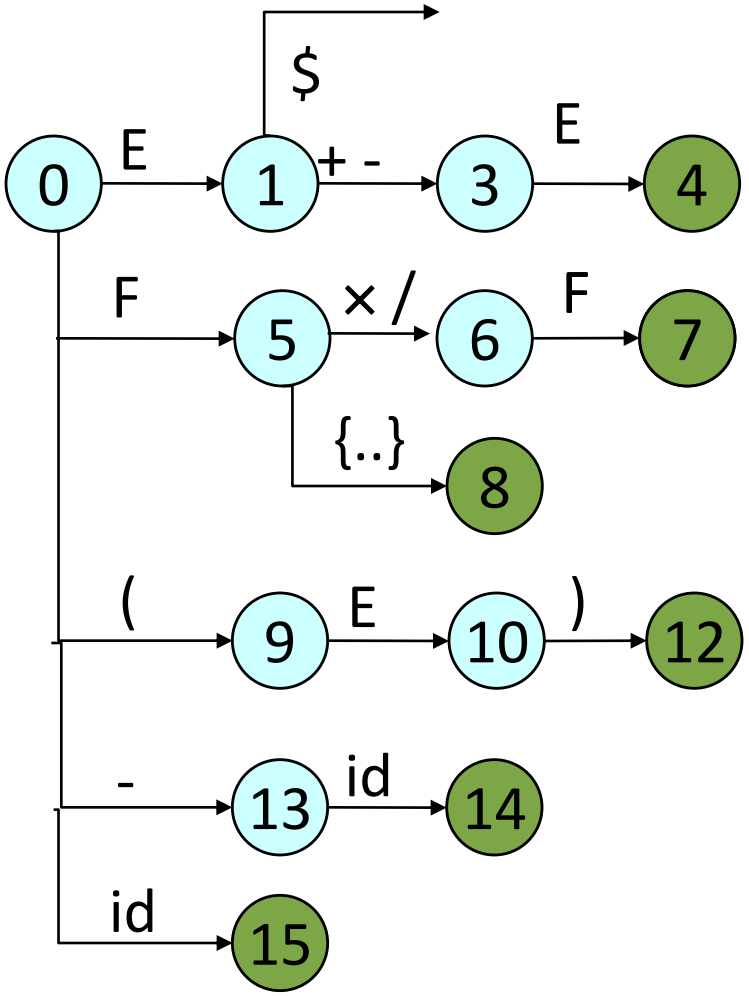
  q:= top(stack); push(input A); /* Dá na vstup A */
- Na vstup sa v každom prípade vracia jediný neterminál. Tento sa v nasledujucom kroku presunie na zásobnik.
- Ak si stav po presune predvypočítame (goto tabuľka) môže vstup byť jednosmerná páska ako pri xLR analýze.

Príklad – kontext senzitívna gramatika

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

Skratky:
addop = {+|-}; mulop = {×|/}
Pravidlo E.2 → F { addop |) | $ } je skratka za množinu pravidiel:
{ E + → F +, E - → F -, E) → F), E $ → F $ }

Skratky okrem toho, že skracujú zápis gramatiky sú aj efektívne implementované, zmenšujú aj rozpoznávací automat.



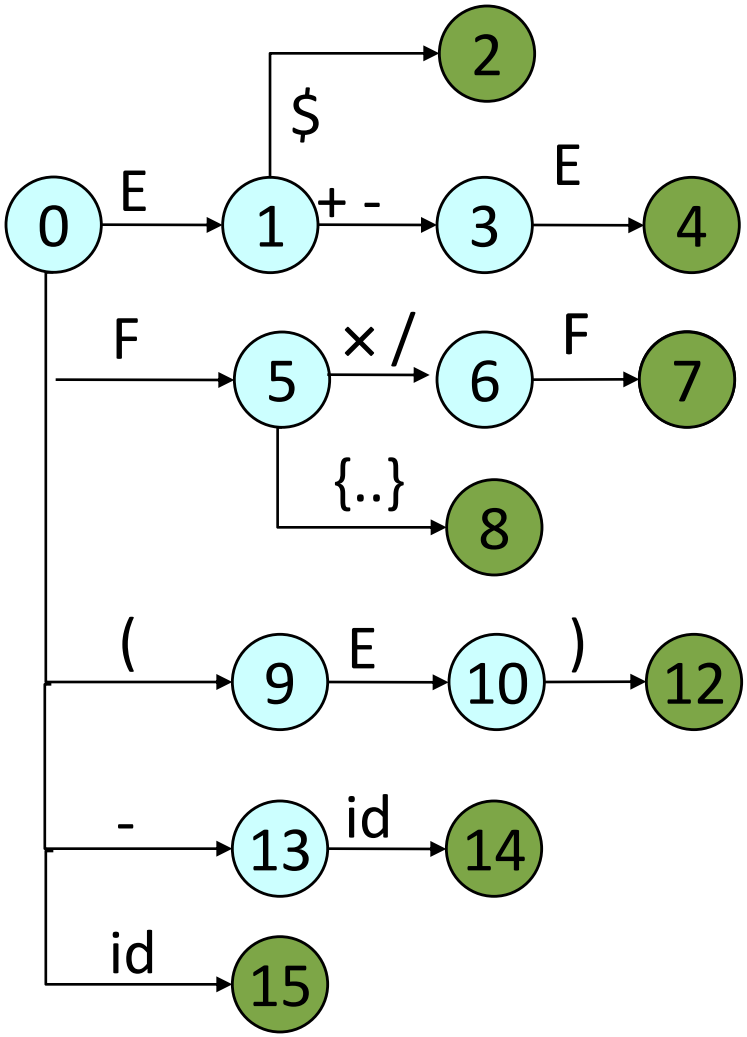
Chyby: diagnostika a recovery

- Pri hľadaní vzorky sa nemôže vyskytnúť chyba. Vzorka sa buď najde, alebo nenajde.
- Ak v trie automatu neexistuje, pokračovanie prejde sa do stavu, ktory zodpovedá rozpoznaniu najdlhšieho suffixu, t.j. v najhoršom prípade do počiatočného stavu.
- Pre syntaktickú analýzu môžeme „nie trie“ prechody doplniť citlivejšie, napr. podľa množín FIRST a Follow.
- 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 $X \Rightarrow r\alpha$.
- Ostatné nie trie hrany sú chybové.
- Chybovým hranám priradíme prechody do chybových stavov (stavov označených zápornými číslami).

Príklad: doplnenie tabuľky prechodov

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	?	

Môžeme aj všetky poslať do stavu 0, bez prečítania vstupu.

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

- Poskytuje aspoň také možnosti spracovania chýb, ako klasické LR automaty.
- Chybové stavy
- Rozpoznanie vzorky sa automaticky zachytí na začiatkoch pravidiel (Panic mode, vstup sa presuvá do zásobniku)
- Relikt (zombie) v zásobníku je vhodnejší pre globálnu opravu a hľadanie minimálnej chyby
- Daň za to je, že sa nemusíme dozvedieť o chybe včas, keď analyzovaný vstup prestane byť prefixom nejakého slova jazyka.

Konkrétny príklad

- aritmetický výraz: id (id+id)$
  LR: E ↑(id+id)$
  SM: FE ↑$
- Čo je lepšie vo všeobecnosti, mi nie je jasné;
  - Prístup LR (kompilátoru): Zistiť chybu najskôr ako sa dá.
  - Prístup SM (hladača vzoriek): Nájsť čo najviac, tak dlhých ako sa dá, odvoditeľných vzoriek.
- Nakoniec to môže dopadnúť rovnako.
  - LR metóda to urobí na dva krát:
    - Panic mode sa zachytí na (a rozpozná sa výraz.
    - Obom prístupom zostane rovnaký relikt v zásobníku a urobia phrase level recovery (missing operand)
  - Pri pozornom ošetrení chýb LR metóda nemusí redukovať (E)

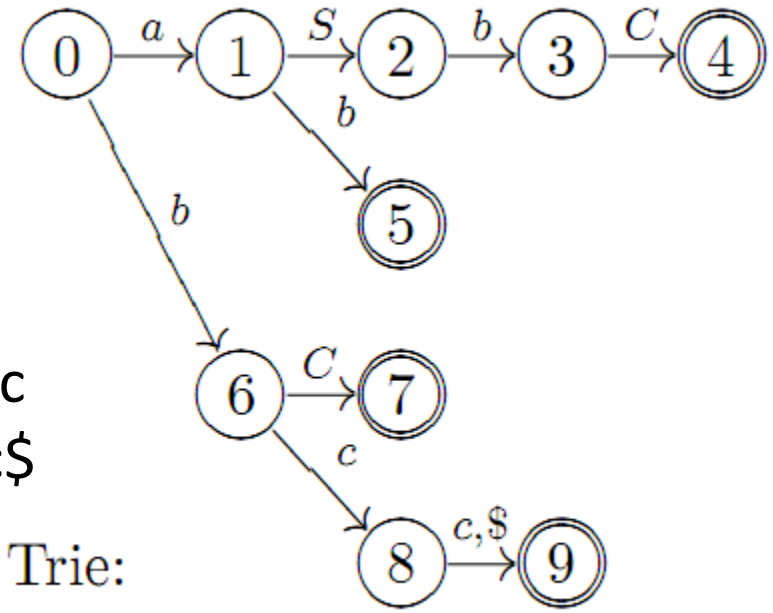
Rozšírenie na gramatiky bez obmedenia – prepisovacie systémy

- Tvar previdiel nie je dôležitý. Hľadáme pravé strany a nahradzujeme ľavými.
- Bez ohraničení na tvar pravidiel sa môže stať prepisy medzi vstupom a zásobníkom sa budú mnohokrát opakovať a analýza nebude lineárna.
- Je to prepisovací systém (Postov, Markovov, ... model algoritmu). Dá sa naprogramovať všetko. Už to nie je o kompilátoroch, ale o programovaní.
- Paradigma syntaxou (pravidlami) riadeného programovania
  - Módne a uživateľský prítuľné, pokiaľ nevznikli grafické prostredia GUI.
  - Vychádzala z lineárneho modelu vstup a výstupu.

Príklad: $\{a^n b^n c^n : n \geq 1\}$

Grammar:

1. S → aSbC
2. aSb → ab
3. Cb → bC
4. bCc → bcc
5. bC$ → 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	$bcc, bc\$$

Kritické je pravidlo 3.
Pre elimináciu jednej trojice abc musí prejsť cez všetky zvyšné b. Spôsobuje, že syntaktická analýza je kvadratickej zložitosti.

Skúste chybný vstup: aaababcbccc
Tu to zistí: aaaSbS ↑Cbcc$;
Tu prestáva byť viable prefix: aaaSb ↑abcbccc$