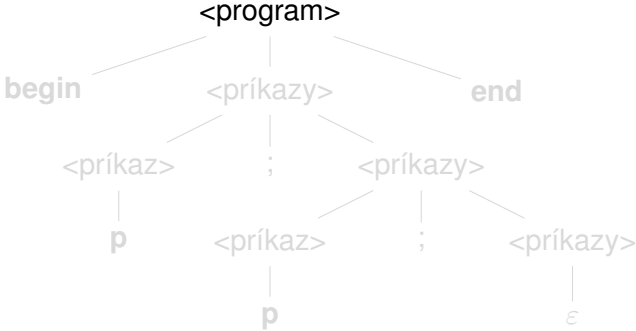
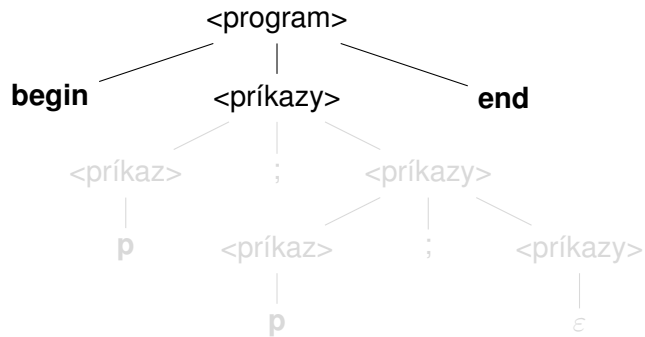


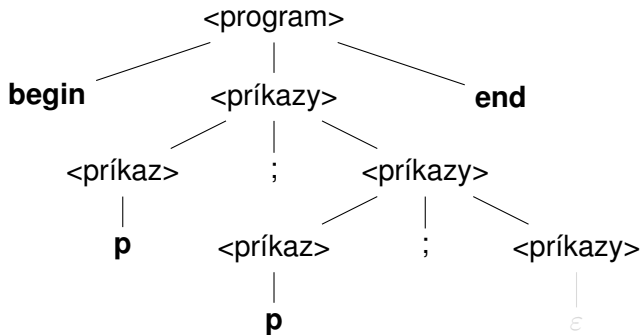
Syntaktický analyzátor

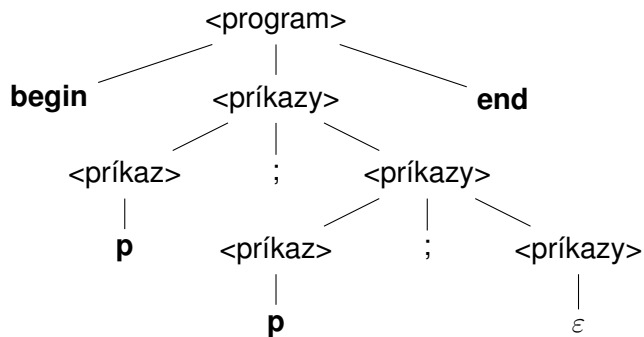
- **Vstupom** syntaktického analyzátoru je **postupnosť lexém** rozpoznaná lexikálnym analyzátorom - tie predstavujú terminálne symboly gramatiky popisujúcej počítačový jazyk.
- **Výstupom** syntaktického analyzátoru je **postupnosť pravidiel**, aplikáciou ktorých dostaneme strom odvodenia programu (vstupného slova), ktorý predstavuje **syntaktickú reprezentáciu** programu.
- Z praktických dôvodov je dobré, aby bol strom odvodenia jednoznačný; navyše musí byť v postupnosti pravidiel jednoznačne určené, na ktorý neterminál sa pravidlo aplikuje (často jednoduché, ak sa povie, že sa jedná o ľavé/pravé odvodenia)











Príklad (pokr.)

Tomu zodpovedá nasledovné odvedenie:

<program> \Rightarrow / **begin**<príkazy>**end** \Rightarrow /
 \Rightarrow / **begin** <príkaz> ; <príkazy> **end** \Rightarrow /
 \Rightarrow / **begin** p ; <príkazy> **end** \Rightarrow /
 \Rightarrow / **begin** p ; <príkaz> ; <príkazy> **end** \Rightarrow /
 \Rightarrow / **begin** p ; p ; <príkazy> **end** \Rightarrow /
 \Rightarrow / **begin** p ; p ; **end**

Top-down analýza

- Uvedený postup hľadania derivačného stromu od koreňa k listom, pričom sa vlastne vytvára ľavá derivácia vstupného slova, sa dá modelovať pomocou nedeterministického zásobníkového automatu.



Teoretický model top-down analyzátoru (pokr.)

- Automat opakuje nasledovné činnosti:
 1. **Expanzia** - ak sa na vrchu zásobníka nachádza neterminál $A \in N$, algoritmus nedeterministicky vyberie jedno z A -pravidiel $A \rightarrow \alpha$ a neterminál A nahradí pravou stranou α .
 2. **Porovnanie** - ak sa na vrchu zásobníka nachádza terminál $a \in T$, porovná sa so symbolom na vstupe t ; ak sa zhodujú, automat odstráni symbol a z vrchu zásobníka a prejde na ďalší vstupný symbol; ak sa nezhodujú, automat sa zasekne a príslušný výpočet nie je akceptačný.
- Automat akceptuje vstupné slovo (=syntakticky správne), ak existuje výpočet, v rámci ktorého sa spracuje celý vstup a automat skončí s prázdny m zásobníkom.
- Ak automat vždy pošle na výstup pravidlo, podľa ktorého vykonal expanziu a zároveň ide o akceptačný výpočet, získa sa postupnosť pravidiel, ktorá dáva ľavú deriváciu slova w v gramatike G .



Syntaktická analýza top-down I

Vstup: Bezkontextová gramatika $G = (N, T, P, S)$, reťazec w

Výstup: Zistenie, či $w \in L(G)$ a postupnosť pravidiel ľavého odvodenia w

- 1: vlož do zásobníka S
- 2: **pokiaľ** zásobník nie je prázdny **rob**
- 3: **ak** na vrchu zásobníka je $A \in N$ **potom**
- 4: odstráň z vrchu zásobníka A
- 5: vlož do zásobníka pravú stranu α niektorého z pravidiel $A \rightarrow \alpha$
- 6: pošli na výstup pravidlo $A \rightarrow \alpha$
- 7: **koniec ak**
- 8: **ak** na vrchu zásobníka je $a \in T$ **potom**
- 9: **ak** a je totožné s aktuálnym vstupným symbolom t **potom**
- 10: odstráň a zo zásobníka
- 11: prejdi na ďalší vstupný symbol t
- 12: **inak**
- 13: neakceptuj vstupné slovo a skončí



Syntaktická analýza top-down II

- 14: **koniec ak**
15: **koniec ak**
16: **koniec pokiaľ**
17: **ak** bolo spracované celé vstupné slovo **potom**
18: akceptuj vstupné slovo
19: **inak**
20: neakceptuj vstupné slovo
21: **koniec ak**



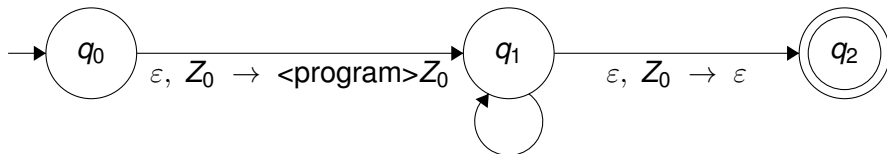
Príklad

Uvažujme gramatiku zo slajdu č. 6 a vetu **begin p ; p ; end**.

r.	Zvyšok vstupu	Zásobník □	Akcia
1	begin p ; p ; end	<program>	E1
2	begin p ; p ; end	begin <príkazy> end	P
3	p ; p ; end	<príkazy> end	E2
4	p ; p ; end	<príkaz>;<príkazy> end	E4
5	p ; p ; end	p ;<príkazy> end	P
6	; p ; end	;<príkazy> end	P
7	p ; end	<príkazy> end	E2
8	p ; end	<príkaz>;<príkazy> end	E4
9	p ; end	p ; <príkazy> end	P
10	; end	; <príkazy> end	P
11	end	<príkazy> end	E3
12	end	end	P
13	ϵ	ϵ	A



Ekvivalentný ZA



$\epsilon, \langle \text{program} \rangle \rightarrow \mathbf{begin}\langle \text{prikazy} \rangle \mathbf{end}$

$\epsilon, \langle \text{prikazy} \rangle \rightarrow \langle \text{prikaz} \rangle ; \langle \text{prikazy} \rangle$

$\epsilon, \langle \text{prikazy} \rangle \rightarrow \epsilon$

$\epsilon, \langle \text{prikaz} \rangle \rightarrow \mathbf{p}$

$\mathbf{begin}, \mathbf{begin} \rightarrow \epsilon$

$\mathbf{end}, \mathbf{end} \rightarrow \epsilon$

$\mathbf{p}, \mathbf{p} \rightarrow \epsilon$

$\mathbf{;}, \mathbf{;} \rightarrow \epsilon$

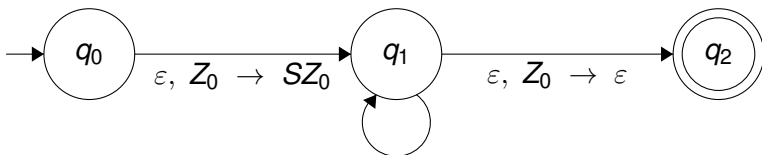


Top-down analýza $\{w \in \{a, b\}^* \mid \#_a(w) = \#_b(w)\}$

Gramatika pre uvedený jazyk:

1. $S \rightarrow SaSbS$
2. $S \rightarrow SbSaS$
3. $S \rightarrow \varepsilon$





$\epsilon, S \rightarrow SaSbS$

$\epsilon, S \rightarrow SbSaS$

$\epsilon, S \rightarrow \epsilon$

$a, a \rightarrow \epsilon$

$b, b \rightarrow \epsilon$



Príklad - výpočet *abaabb*

Uvažujme gramatiku zo slajdu č. 16 a reťazec *abaabb*. Potom akceptujúci výpočet ZA bude (zápis vo forme tabuľky):

r.	Zvyšok vstupu	Zásobník □	Akcia
1	<i>abaabb</i>	S	E1
2	<i>abaabb</i>	SaSbS	E3
3	<i>abaabb</i>	aSbS	P
4	<i>baabb</i>	SbS	E3
5	<i>baabb</i>	bS	P
6	<i>aabb</i>	S	E1
7	<i>aabb</i>	SaSbS	E3
8	<i>aabb</i>	aSbS	P
9	<i>abb</i>	SbS	E1
10	<i>abb</i>	SaSbSbS	E3
11	<i>abb</i>	aSbSbS	P
12	<i>bb</i>	SbSbS	E3



Príklad - výpočet *abaabb*

r.	Zvyšok vstupu	Zásobník □	Akcia
13	<i>bb</i>	<i>bSbS</i>	P
14	<i>b</i>	<i>SbS</i>	E3
15	<i>b</i>	<i>bS</i>	P
16	ϵ	<i>S</i>	E3
17	ϵ	ϵ	A



Príklad - výpočet *abb*

r.	Zvyšok vstupu	Zásobník \square	Akcia
1	<i>abb</i>	<i>S</i>	E1
2	<i>abb</i>	<i>SaSbS</i>	E3
3	<i>abb</i>	<i>aSbS</i>	P
4	<i>bb</i>	<i>SbS</i>	E3
5	<i>bb</i>	<i>bS</i>	P
6	<i>b</i>	<i>S</i>	E2
7	<i>b</i>	<i>SbSaS</i>	E3
8	<i>b</i>	<i>bSaS</i>	P
9	ε	<i>SaS</i>	E3
10	ε	<i>aS</i>	ERROR

Pre *abb* by aj všetky iné možné výpočty skončili s chybou - slovo *abb* teda nepatrí do $L(G)$.



Vlastnosti teoretického modelu

- Predstavený model je použiteľný s ľubovoľnou bezkontextovou gramatikou. Dokáže nájsť derivačný strom pre slová generované aj nejednoznačnými, aj nedeterministickými gramatikami.
- **Nedeterminizmus** teoretického modelu je však prekážkou pri jeho praktickej použiteľnosti, pretože pri hľadaní derivácie / spracovaní slova je potrebné sledovať všetky možné vetvy výpočtu.
 - V príklade na slajde 14 je nedeterminizmus na riadkoch 3,7,11, kde sa vyberalo z 2 pravidiel na expanziu neterminálu <príkazy>.
- Ako teda vytvoriť prakticky použiteľný (t.j. deterministický) top-down parser?



Syntaktický analyzátor jazyka LL(1)

Aby bol syntaktický analyzátor použiteľný v praxi, je potrebné, aby vedel **deterministicky** spracovať vstupné reťazce s **lineárnou** časovou zložitou.

T.j. musíme odstrániť nedeterminizmus z teoretického modelu.

Daný model analyzátoru obsahuje jediný nedeterminizmus - keď je potrebné expandovať neterminál A na vrchu zásobníka a príslušná gramatika obsahuje viacero pravidiel s neterminálom A na ľavej strane.

V praxi sa nedeterminizmus odstráni tak, že automat si príslušné pravidlo **vyberie** na základe **aktuálneho vstupného symbolu** t .

Takýto analyzátor má názov LL(1) = left-left-1. T.j. vstupný reťazec sa spracúva **zľava**, vytvára sa **ľavé** odvodenie a na rozhodovanie sa používa 1 vstupný symbol.

Syntaktický analyzátor jazyka LL(1)

Definícia

Nech $G = (N, T, P, S)$ je bezkontextová gramatika. **Rozkladová tabuľka syntaktického analyzátoru LL(1)** $RT[A, t]$ je zobrazenie $N \times T_\epsilon \rightarrow P$ popisujúce, ktoré pravidlo sa použije na expanziu neterminálu A nachádzajúceho sa na vrchu zásobníka, ak aktuálny vstupný symbol je t (ak $t = \epsilon$, potom bol prečítaný celý vstupný reťazec).

Rozkladová tabuľka je zvyčajne neúplné zobrazenie a pre danú gramatiku môžu existovať kombinácie $RT[A, t]$, ktoré nemajú priradené pravidlo.



Syntaktický analyzátor jazyka LL(1) I

Vstup: Bezkontextová gramatika $G = (N, T, P, S)$, reťazec w , rozkladová tabuľka syntaktického analyzátoru LL(1) RT

Výstup: Zistenie, či $w \in L(G)$ a postupnosť pravidiel ľavého odvodu w

- 1: vlož do zásobníka S
- 2: $t \leftarrow$ prvá lexéma zo vstupu
- 3: **pokiaľ** zásobník nie je prázdny **rob**
- 4: $X \leftarrow$ symbol z vrchu zásobníka a zároveň ho odstráň zo zásobníka
- 5: **ak** $X \in N$ a zároveň $RT[X, t] = X \rightarrow \alpha$ **potom**
- 6: vlož do zásobníka α aby ľavý krajný symbol bol na vrchu
- 7: pošli na výstup pravidlo $X \rightarrow \alpha$
- 8: **inak**
- 9: **ak** $X \in T$ a zároveň $X = t$ **potom**
- 10: $t \leftarrow$ ďalšia lexéma zo vstupu
- 11: **inak**
- 12: spracuj syntaktickú chybu



Syntaktický analyzátor jazyka LL(1) II

- 13: **koniec ak**
- 14: **koniec ak**
- 15: **koniec pokiaľ**
- 16: akceptuj vstupné slovo



Ako vytvoriť rozkladovú tabuľku - idea

- Ak je na vrchu zásobníka neterminál A a na vstupe terminál t , pýtame sa, za akých podmienok použiť pravidlo $A \rightarrow \alpha$, aby sa analyzátor nezasekol.
- Analyzátor (zásobníkový automat) sa zasekne v momente, keď sa vstupný terminál nezhoduje s terminálom na vrchu zásobníka počas porovnania.
- Preto pri výbere pravidla do RT postupujeme tak, aby sme mali zabezpečené, že po expanzii neterminálu A zvoleným pravidlom $A \rightarrow \alpha$ bude vstupný symbol t rovný terminálu, ktorý sa môže zjaviť na vrchu zásobníka.
- Ak expandujeme podľa pravidla $A \rightarrow \alpha$, tak na vrchu zásobníka sa môže zjaviť len terminál patriaci do množiny:
 - $FIRST(\alpha) - \{\epsilon\}$
 - ak $\epsilon \in FIRST(\alpha)$ tak $FOLLOW(A)$.



Ako vytvoriť rozkladovú tabuľku - množiny *PREDICT*

Definícia

Nech $G = (N, T, P, S)$ je redukovaná gramatika a $A \rightarrow \alpha \in P$. Potom:

$$\begin{aligned} PREDICT(A \rightarrow \alpha) &= \\ &= \begin{cases} FIRST(\alpha), & ak \ \varepsilon \notin FIRST(\alpha) \\ (FIRST(\alpha) - \{\varepsilon\}) \cup FOLLOW(A), & ak \ \varepsilon \in FIRST(\alpha) \end{cases} \end{aligned}$$

T.j. množina *PREDICT* signalizuje, ktoré pravidlo použiť na expanziu A .
Samozrejme, ak sa máme **jednoznačne** rozhodnúť, musí platiť: Ak $A \rightarrow \alpha_i$ a $A \rightarrow \alpha_j$ sú 2 rôzne A -pravidlá, potom:

$$PREDICT(A \rightarrow \alpha_i) \cap PREDICT(A \rightarrow \alpha_j) = \emptyset$$

Definícia

*Redukovaná gramatika $G = (N, T, P, S)$ v ktorej všetky dvojice pravidiel s rovnakou ľavou stranou majú prázdny prienik množín $PREDICT$, sa nazýva **LL(1) gramatika**.*

Ak je teda gramatika typu $LL(1)$, potom RT zostrojíme nasledovne:

Ak $t \in PREDICT(A \rightarrow \alpha)$, potom $RT[A, t]$ obsahuje pravidlo $A \rightarrow \alpha$.

Ak je gramatika typu $LL(1)$, každá položka rozkladovej tabuľky obsahuje maximálne jedno pravidlo a rozhodovanie o použití pravidla pri expanzii je jednoznačné.



Príklad zostrojenia RT

Zostrojte RT pre gramatiku:

1. $\langle \text{program} \rangle \rightarrow \mathbf{begin} \langle \text{príkazy} \rangle \mathbf{end}$
2. $\langle \text{príkazy} \rangle \rightarrow \langle \text{príkaz} \rangle ; \langle \text{príkazy} \rangle$
3. $\langle \text{príkazy} \rangle \rightarrow \varepsilon$
4. $\langle \text{príkaz} \rangle \rightarrow \mathbf{p}$

Najprv zostrojíme množiny *FIRST* a *FOLLOW*:

FIRST	$\langle \text{program} \rangle$	$\langle \text{príkazy} \rangle$	$\langle \text{príkaz} \rangle$	begin	end	;	p
Celkom	begin	p, ε	p	begin	end	;	p

FOLLOW	$\langle \text{program} \rangle$	$\langle \text{príkazy} \rangle$	$\langle \text{príkaz} \rangle$
Celkom:	ε	end	;



Príklad zostrojenia RT (pokr.)

Výsledné množiny *PREDICT* pre pravidlá:

- $PREDICT(1.) = FIRST(\mathbf{begin}\langle\text{príkazy}\rangle\mathbf{end}) = \{\mathbf{begin}\}$.
- $PREDICT(2.) = FIRST(\langle\text{príkaz}\rangle;\langle\text{príkazy}\rangle) = \{\mathbf{p}\}$
- $PREDICT(3.) = FOLLOW(\langle\text{príkazy}\rangle) = \{\mathbf{end}\}$
- $PREDICT(4.) = FIRST(\mathbf{p}) = \{\mathbf{p}\}$.

T.j. rozkladová tabuľka:

RT	begin	end	;	p	ϵ
$\langle\text{program}\rangle$	1.				
$\langle\text{príkazy}\rangle$		3.		2.	
$\langle\text{príkaz}\rangle$				4.	

Keďže každá položka obsahuje max. 1 pravidlo, gramatika je naozaj LL(1) gramatika.



Činnosť $LL(1)$ -analyzátoru pre vstup **begin p ; end**.

r.	Zvyšok vstupu	Zásobník □	Akcia
1	begin p ; end	<program>	E1
2	begin p ; end	begin<príkazy>end	P
3	p ; end	<príkazy> end	E2
4	p ; end	<príkaz>;<príkazy> end	E4
5	p ; end	p ;<príkazy>end	P
6	; end	;<príkazy> end	P
7	end	<príkazy> end	E3
8	end	end	P
9	ϵ	ϵ	A



Činnosť $LL(1)$ -analyzátoru pre vstup **begin p end**.

r.	Zvyšok vstupu	Zásobník □	Akcia
1	begin p end	<program>	E1
2	begin p end	begin <príkazy> end	P
3	p end	<príkazy> end	E2
4	p end	<príkaz>;<príkazy> end	E4
5	p end	p ;<príkazy> end	P
6	end	;<príkazy> end	SYNTAX ERROR

Vidíme, že počas výpočtu nastala situácia, keď na vrchu zásobníka bol terminál ; avšak na vstupe terminál **end**. To znamená, že počas operácie porovnania nastane chyba signalizujúca **syntaktickú chybu**, teda reťazec **begin p end** v gramatike **nemá deriváciu**.



Činnosť $LL(1)$ -analyzátoru pre vstup **p ; end**.

r.	Zvyšok vstupu	Zásobník □	Akcia
1	p ; end	<program>	SYNTAX ERROR

Vidíme, že počas výpočtu nastala situácia, keď na vrchu zásobníka bol neterminál <program> a na vstupe terminál **p**. Keďže v rozkladovej tabuľke **neexistuje** pridelené pravidlo pre kombináciu $RT[< program >, \mathbf{p}]$, znamená to, že v gramatike nie je možné odvodiť z neterminálu <program> reťazec začínajúci symbolom **p** a teda došlo k **syntaktickej chybe**, pretože vstupný reťazec určite v gramatike **nemá deriváciu**.



Príklad zostrojenia RT

Zostrojte RT pre gramatiku:

1. $S \rightarrow SaSbS$
2. $S \rightarrow SbSaS$
3. $S \rightarrow \varepsilon$

Najprv zostrojíme množiny *FIRST* a *FOLLOW*:

FIRST	S	a	b
Celkom	$\{\varepsilon, a, b\}$	$\{a\}$	$\{b\}$

FOLLOW	S
Celkom:	$\{\varepsilon, a, b\}$



Príklad zostrojenia RT (pokr.)

Výsledné množiny *PREDICT* pre pravidlá:

- $PREDICT(1.) = FIRST(SaSbS) = \{a, b\}$.
- $PREDICT(2.) = FIRST(SbSaS) = \{a, b\}$
- $PREDICT(3.) = (FIRST(\epsilon) \setminus \{\epsilon\}) \cup FOLLOW(S) =$
 $= FOLLOW(S) = \{a, b, \epsilon\}$

T.j. rozkladová tabuľka:

RT	a	b	ϵ
S	1., 2., 3.	1., 2., 3.	3.

Keďže existujú položky s viac ako 1 pravidlom, gramatika **nie je** LL(1) gramatika.



Poznámky

- $LL(1)$ analyzátory/jazyky sa dajú rozšíriť na rozhodovanie o expanzii na základe k vstupných symbolov, $k \geq 1$. Vtedy hovoríme o $LL(k)$ gramatikách/analyzátoroch.
- Problémom je, že pre niektoré bezkontextové gramatiky sa nedá zostrojiť ani $LL(1)$ analyzátor, ani $LL(k)$ analyzátor.
- V niektorých prípadoch je riešením pokúsiť sa nájsť ekvivalentnú gramatiku pre daný jazyk, ktorá $LL(1)$ je.



Príklad

Gramatika, ktorá generuje jazyk $\{w \in \{a, b\}^* \mid \#_a(w) = \#_b(w)\}$:

1. $S \rightarrow SaSbS$
2. $S \rightarrow SbSaS$
3. $S \rightarrow \varepsilon$

nie je $LL(1)$



Ten istý jazyk, t.j. $\{w \in \{a, b\}^* \mid \#_a(w) = \#_b(w)\}$, sa však predsa len dá generovať $LL(1)$ gramatikou, konkrétne:

1. $S \rightarrow aBS \mid bAS \mid \varepsilon$
2. $A \rightarrow a \mid bAA$
3. $B \rightarrow b \mid aBB$



Rozkladová tabuľka danej gramatiky

Daná gramatika pre if-then-else nie je $LL(1)$ gramatika, pretože:

RT	if	then	else	iný	p	ϵ
<príkaz>	1.			2.		
<else časť>			<u>3.</u> 4.			4.
<podmienka>					5.	

Keďže väčšinou sa `else` viaže s najbližším predošlým príkazom `if`, tak sa dá pravidlu č. 3 prednosť pred pravidlom číslo 4, ak nastane situácia, že na vrchu zásobníka je symbol <else časť> a na vstupe terminál `else`.



