

Vkládaný symbol do zásobníku

	a	b	c	f	A	B	C	S
a								
b			c				C	
c								
f								
A		b				B		
B								
C								
S								
#	a			f	A			S

vrchol zásobníku tab. g =

Tak snadné to je jen u tzv. triviálních gramatik (jaké musí mít vlastnosti?)
 Mohou mít rekurzivní pravidla?
 Mohou vůbec obsahovat rekurzivní symboly?

Vytvoření tabulky akcí a tabulky přechodů pro triviální LR gramatiku

1. Tabulka akcí f , (řádky jsou z $N \cup T \cup \{ \# \}$)
 - a) Je-li symbol $X \in N \cup T$ na konci pravidla $i: A \rightarrow \alpha$, pak $f(X) = \text{redukce}(i)$,
 - b) $f(S) = \text{přijetí}$
 - c) $f(X) = \text{přesun}$ v ostatních případech
2. Tabulka přechodů g (sloupce jsou $N \cup T$, řádky jsou $N \cup T \cup \{ \# \}$)
 - a) $g(\#, X) = X$ jestliže v $G \exists$ derivace $S \Rightarrow^* X \alpha$
 - b) $g(X, Y) = Y$ jestliže G obsahuje pravidlo $A \rightarrow \alpha X Y \beta$
 - c) $g(X, Y) = Y$ jestliže G obsahuje pravidlo $A \rightarrow \alpha X B \beta$, kde $B \in N$ a $B \Rightarrow^+ Y \gamma$
 - d) $g(X, Y) = \text{chyba}$ v ostatních případech

Algoritmus SA triviální LR gramatiky (platí i pro LR(0))

Označme symbol na vrcholu zásobníku X, dno označme #.

1.
 - a. Je-li $f(X) = \text{přesun}$, přečti vstupní symbol a jdi na 2.
 - b. Je-li $f(X) = \text{redukce}(i)$, vyloučí se ze zásobníku pravá strana pravidla i , číslo i se přidá do výstupu a přejde se na bod 2.
 - c. Je-li $f(X) = \text{přijetí}$, pak při zároveň prázdném vstupním řetězci ukončíme činnost akceptací, při neprázdném ukončíme odmítnutím.
2.

Je-li Y symbol , který má být vložen do zásobníku, provedeme:

 - a. Je-li $g(X, Y) = Z$, uložíme Z na vrchol zásobníku a opakujeme 1.
 - b. Je-li $g(X, Y) = \text{chyba}$, ukončíme analýzu chybou.

Konfiguraci zapisujeme ve tvaru

(**obsah zásob. s vrcholem vpravo, zbytek vst. řetězce**, čísla pr. pro redukce)

Je to názornější = **Tvoří větnou formu**

Př. na tabuli analýza řetězce dle tabulek pro $G_{10}[S]$

(#, fbc, -)		(#f, bc, -)
		(#A, bc, 2)
		(#Ab, c, 2)
		(#Abc, e, 2)
		(#AbC, e, 25)
		(#AB, e, 254)
		(#S, e, 2541)

LR(0) gramatiky

Při násobném výskytu některého symbolu na pravé straně pravidel, je nutné rozlišovat (třeba indexem) tyto výskyty i v procesu SA, tedy i v zásobníku. Pro nekomplikované G to zvládneme jako u triviálních G .

Př. $G_{11}[S]$	1	$S \rightarrow B$		$S \rightarrow B1$
	2	$B \rightarrow a B b$		$B \rightarrow a B2 b1$
	3	$B \rightarrow A$		$B \rightarrow A1$
	4	$A \rightarrow b A$		$A \rightarrow b2 A2$
	5	$A \rightarrow c$		$A \rightarrow c$

Sestrojíme na tabuli f a g (pro G_{11} to ještě zvládneme algoritmem pro triviální gramatiku)

Zás	akce	a	b	c	B	A	S
#	přesun	a	b2	c	B1	A1	S
a	přesun	a	b2	c	B2	A1	
b1	R2						
b2	přesun		b2	c		A2	
c	R5						
B1	R1						
B2	přesun		b1				
A1	R3						
A2	R4						
S	akcept						

Př. syntaktické analýzy na tabuli

(#, abcb, -)	(#a, bcb, -)
	(#ab ₂ , cb, -)
	(#ab ₂ c, b, -)
	(#ab ₂ A ₂ , b, 5)
	(#aA ₁ , b, 5 4)
	(#aB ₂ , b, 5 4 3)
	(#a B ₂ b ₁ , e, 5 4 3)
	(# B ₁ , e, 5 4 3 2)
	(#S, e, 5 4 3 2 1)

U komplikovanějších gramatik použijeme místo indexování symbolů k výpočtu tabulek tzv. množiny položek.

Výpočet rozkladových LR tabulek pomocí množin položek

Platí, že $g(\#, X) = X$ jestliže v $G \exists$ derivace $S \Rightarrow^* X \alpha$

Podle dosavadního postupu

proto $g(\#, S) =$	S	}	symboly, které mohou být v zásobníku přímo u #
$g(\#, B) =$	B1		
$g(\#, a) =$	a		
$g(\#, A) =$	A1		
$g(\#, c) =$	c		
$g(\#, b) =$	b2		

$f(\#) =$ přesun

Musíme zjistit jaké situace v konfiguracích mohou při SA nastávat (co lze kdy vkládat do zásobníku) a jaká akce je ta jediná správná.

Situace charakteristická pro určitý vrcholový symbol zásobníku je popsatelná tzv. množinou LR(0) položek.

Pro # na vrcholu to je

vrchol zásobníku	ještě venku = nezpracováno
#	B
:	.
S	→
B	→
B	→
A	→
A	→

$S \rightarrow . B$
 $B \rightarrow . a B b$
 $B \rightarrow . A$
 $A \rightarrow . b A$
 $A \rightarrow . c$

Tečka symbolizuje rozhraní *zásobník . dosud nezpracovaná část vstupu*

Vkládání symbolů do zásobníku (přesouváním ze vstupu nebo redukcemi vrcholového řetězce) je symbolizováno posouváním tečky.

Z množiny pro # plyne, že při vrcholu # mohou k němu vložit B (ale v jaké variantě?) nebo a nebo A (ale v jaké variantě?) nebo b (ale v jaké variantě?) nebo c.

Posouváním tečky dostáváme postupně konečný soubor množin položek a současně i graf přechodů mezi množinami. Tato informace plně popisuje možné stavy při SA.

Algoritmus

Vytvoření souboru množin $LR(0)$ položek.

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

Výstup: Soubor \varnothing množin $LR(0)$ položek pro G .

Metoda:

- Počáteční množinu $LR(0)$ položek M_0 vytvoříme takto:
 - $M_0 = \{S \rightarrow \cdot \alpha : S \rightarrow \alpha \in P\}$.
 - Jestliže $A \rightarrow \cdot B \alpha \in M_0, B \in N$ a $B \rightarrow \beta \in P$, pak
 $M_0 = M_0 \cup \{B \rightarrow \cdot \beta\}$.
 - Opakujeme krok b) tak dlouho, dokud je možné přidávat nové položky do M_0 .
 - $\varnothing = \{M_0\}$, M_0 je počáteční množina.
- Jestliže jsme zkonstruovali množinu $LR(0)$ položek M_i , zkonstruujeme pro každý symbol $X \in N \cup T$ takový, že leží v některé $LR(0)$ položce v M_i za tečkou další množinu $LR(0)$ položek M_j , kde j je index větší než nejvyšší index dosud vytvořené množiny M_i takto:
 - $M_j = \{A \rightarrow \alpha X \cdot \beta : A \rightarrow \alpha \cdot X \beta \in M_i\}$.
 - Jestliže $A \rightarrow \alpha \cdot B \beta \in M_j, B \in N, B \rightarrow \gamma \in P$, pak
 $M_j = M_j \cup \{B \rightarrow \cdot \gamma\}$.
 - Opakujeme krok (b) tak dlouho, dokud je možné do M_j přidávat nové položky.
 - $\varnothing = \varnothing \cup \{M_j\}$.
- Krok 2) opakujeme pro všechny vytvořené množiny M_j , dokud je možné do \varnothing přidávat nové množiny M_j .

Poznámka: Krok 1a) a 2a) budeme nazývat vytvoření základů množin $LR(0)$ položek, opakované provádění kroku 1b) a 2b) nazveme vytváření uzávěrů množin položek.

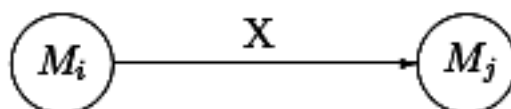
Definice

$GOTO(M_i, X) = M_j$, jestliže položky tvaru $A \rightarrow \alpha \cdot X \beta$ leží v M_i a základ množiny M_j byl vytvořen položkami tvaru $A \rightarrow \alpha X \cdot \beta$.

Funkce GOTO si můžeme znázornit jako orientovaný hranově a uzlově ohodnocený graf

takto:

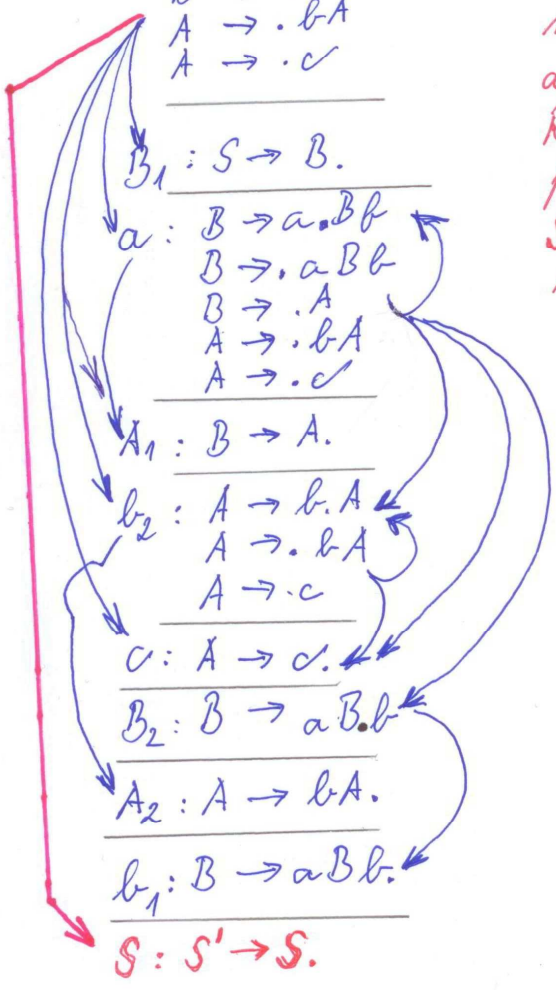
Funkci $GOTO(M_i, X) = M_j$ bude odpovídat graf



Př. na tabuli (zabere téměř stránku)

- G: 1) $S \rightarrow B$
 2) $B \rightarrow aBb$
 3) $B \rightarrow A$
 4) $A \rightarrow bA$
 5) $A \rightarrow c$

#: $S' \rightarrow \cdot S$
 $S \rightarrow \cdot B$
 $B \rightarrow \cdot aBb$
 $B \rightarrow \cdot A$
 $A \rightarrow \cdot bA$
 $A \rightarrow \cdot c$



Když vyplníme podle této metody množiny množin položek tabulky f a g, bude nám chybět údaj do řádku a sloupce pro S. Rozšíříme proto gramatiku o pravidlo $S' \rightarrow S$, kde S' bude nový počáteční symbol a doplníme graf

$C[2] : A \rightarrow B$

Algoritmus

Konstrukce tabulky akcí f a tabulky přechodů g pro LR(0) gramatiku

Vstup: Soubor LR(0) položek

Výstup: Tabulky f, g

Postup:

1) Řádky f budou odpovídat množinám položek. Sestrojí se následovně:

Je-li v množině položek M obsažena položka tvaru $A \rightarrow \alpha$.
pak $f(M) = \text{redukce}(i)$, kde i je číslo pravidla $A \rightarrow \alpha$

Je-li v množině položek X obsažena položka tvaru $S' \rightarrow S$.
pak $f(M) = \text{přijetí}$

$f(M) = \text{přesun}$ v ostatních případech

2) Tabulka přechodů g odpovídá přechodové funkci GOTO mezi množinami položek. Řádky g budou odpovídat množinám položek. Sestrojí se následovně:

a) Je-li $\text{GOTO}(M_i, X) = M_j$, pak $g(M_i, X) = M_j$

b) Je-li $\text{GOTO}(M_i, X) = \text{prázdná množina}$, pak $g(M_i, X) = \text{chyba}$

Př. Konstruujeme f a g na základě LR(0) položek. Všimněte si, něco v nich chybí. To je důvod použití rozšířené gramatiky (viz doplnění předchozího grafu přechodů o červenou část)

Zás	akce	a	b	c	B	A	S
#	přesun	a	b2	c	B1	A1	S
a	přesun	a	b2	c	B2	A1	
b1	R2						
b2	přesun		b2	c		A2	
c	R5						
B1	R1						
B2	přesun		b1				
A1	R3						
A2	R4						
S	akcept						

SLR(k) gramatiky (Simple LR(k))

Obsahuje-li některá z množin LR(0) položek

jak položku	$A \rightarrow \alpha \cdot$	}	(tzv. konflikt redukce redukce)
tak položku	$B \rightarrow \beta \cdot$		
či položku	$C \rightarrow \gamma \cdot \delta$		

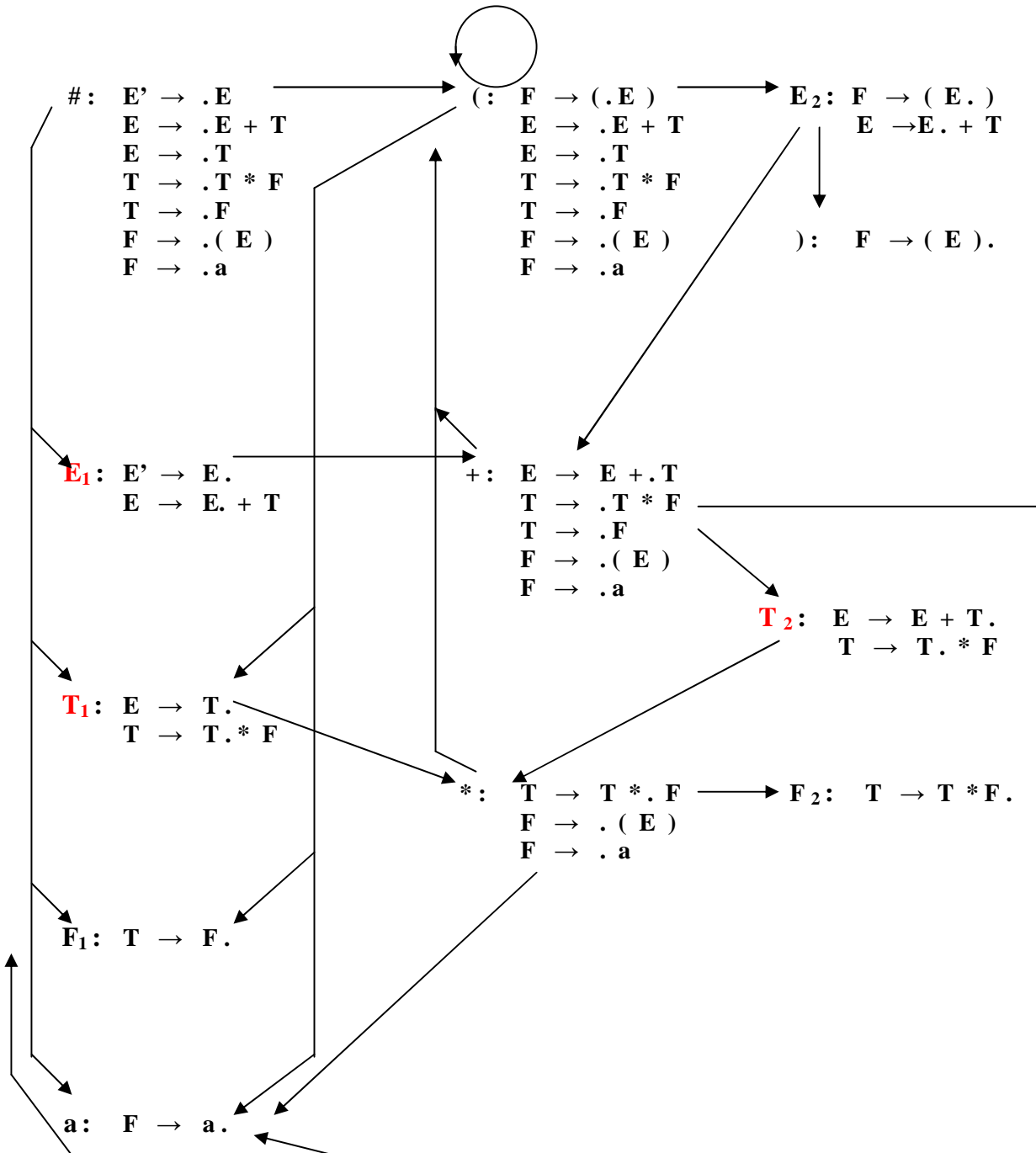
pak gramatika není LR(0). Pokud to půjde napravit prohlížením vstupujících symbolů, pak se jedná o některou z podtříd LR(k) gramatik.

Př. $G_{12}[E']$

0	E'	\rightarrow	E
1	E	\rightarrow	$E + T$
2	E	\rightarrow	T
3	T	\rightarrow	$T * F$
4	T	\rightarrow	F
5	F	\rightarrow	(E)
6	F	\rightarrow	a

Vytvoříme množiny LR(0) položek (na tabuli, téměř stránka)

LR(0) množiny položek pro G[E]



V červeně označených množinách položek nastal konflikt redukce přesun

Předpokládejme, že $A \rightarrow \alpha \cdot \beta$ a $B \rightarrow \gamma \cdot \delta$ jsou dvě různé položky z M . Jestliže každé dvě takové položky splňují alespoň jednu z podmínek:

1. Ani β , ani δ nejsou prázdné řetězy,
2. $\beta \neq e$, $\delta = e$ a $\text{FOLLOW}_k(B) \cap \text{FIRST}_k(\beta \text{FOLLOW}_k(A)) = \emptyset$ pro $\beta \in T(N \cup T)^*$,
3. $\beta = e$, $\delta \neq e$ a $\text{FOLLOW}_k(A) \cap \text{FIRST}_k(\delta \text{FOLLOW}_k(B)) = \emptyset$ pro $\delta \in T(N \cup T)^*$
4. $\beta = \delta = e$ a $\text{FOLLOW}_k(A) \cap \text{FOLLOW}_k(B) = \emptyset$,

pak G se nazývá jednoduchá $LR(k)$ gramatika (zkráceně $SLR(k)$ gramatika). Zápis $\beta \in T(N \cup T)^*$ znamená, že řetěz β začíná terminálním symbolem.

Pro $SLR(k)$ gramatiky můžeme sestrojít tabulku akcí f a tabulku přechodů g pomocí následujícího algoritmu.

Algoritmus

Konstrukce tabulky akcí f a tabulky přechodů g pro $SLR(k)$ gramatiku

Vstup: $SLR(k)$ gramatika $G = (N, T, P, S)$ a soubor množin $LR(0)$ položek φ pro G .

Výstup: Tabulka akcí f a tabulka přechodů g pro G .

Metoda:

1. Tabulka akcí f bude mít řádky označeny stejně jako množiny položek z φ . Sloupce tabulky budou označeny řetězy symbolů z T^{*k} ,
 - (a) $f(M_i, u) = \text{přesun}$, jestliže $A \rightarrow \beta_1 \cdot \beta_2 \in M_i$, $\beta_2 \in T(N \cup T)^*$ a $u \in \text{FIRST}_k(\beta_2 \text{FOLLOW}_k(A))$,
 - (b) $f(M_i, u) = \text{redukce}(j)$, jestliže $j \geq 1$ a $A \rightarrow \beta \cdot \in M_i$, $A \rightarrow \beta$ je j -té pravidlo v P a $u \in \text{FOLLOW}_k(A)$,
 - (c) $f(M_i, e) = \text{přijetí}$, jestliže $S' \rightarrow S \cdot \in M_i$,
 - (d) $f(M_i, u) = \text{chyba}$ ve všech ostatních případech.
2. Tabulka přechodů g odpovídá funkci GOTO.
 - (a) Je-li $\text{GOTO}(M_i, x) = M_j$, kde $x \in (N \cup T)$, pak $g(M_i, x) = M_j$.
 - (b) Je-li $\text{GOTO}(M_i, x)$ prázdná množina pro $x \in (N \cup T)$, pak $g(M_i, x) = \text{chyba}$.

Př. na tabuli cca 15 řádek

Algoritmus

Syntaktická analýza pro $SLR(k)$ gramatiky (algoritmus je použitelný i pro $LALR(k)$ gramatiky a pro $LR(k)$ gramatiky – viz. dále)

Vstup: Tabulka akcí f a tabulka přechodů g pro $G = (N, T, P, S)$, vstupní řetěz $w \in T^*$ a počáteční symbol zásobníku M_0 (označení počáteční množiny $LR(0)$ položek).

Výstup: Pravý rozklad v případě, že $w \in L(G)$, jinak chybová indikace.

Metoda: Algoritmus čte symboly ze vstupního řetězu w , využívá zásobník a vytváří řetěz čísel pravidel, která byla použita při redukcích. V zásobníku je na začátku symbol M_0 .

Opakujeme kroky 1), 2), a 3) dokud nenastane *přijat* nebo *chyba*.

Symbol X je symbolem na vrcholu zásobníku.

1. Určíme řetěz k prvních symbolů z dosud nepřečtené části vstupního řetězu a označíme jej u .
2. (a) Je-li $f(X, u) = \text{přesun}$, přečte se vstupní symbol a přejdeme na krok 3).
(b) Je-li $f(X, u) = \text{redukce}(i)$, vyloučíme ze zásobníku tolik symbolů, kolik je symbolů na pravé straně i -tého pravidla $(i)A \rightarrow \alpha$ a do výstupního řetězu připojíme číslo pravidla (i) . Přejdeme na krok 3).
(c) Je-li $f(X, e) = \text{přijetí}$, ukončíme analýzu a výstupní řetěz je pravý rozklad vstupní věty w v případě, že vstupní řetěz je celý přečten, jinak chyba.
(d) Je-li $f(X, u) = \text{chyba}$, ukončíme rozklad chybovou indikací.
3. Je-li Y symbol, který má být uložen do zásobníku (přečtený symbol ve 2a) nebo levá strana pravidla použitého při redukci (v 2b)) a X je symbol na vrcholu zásobníku, pak:
(a) Je-li $g(X, Y) = Z$, pak uložíme Z na vrcholu zásobníku a opakujeme od kroku 1.
(b) Je-li $g(X, Y) = \text{chyba}$, ukončíme analýzu chybovou indikací.

Konfigurací algoritmu budeme rozumět trojici:

(α, x, π) , kde α je obsah zásobníku
 x je dosud nepřečtená část vstupního řetězce textu,
 π je dosud vytvořená část výstupního řetězce pravidel.

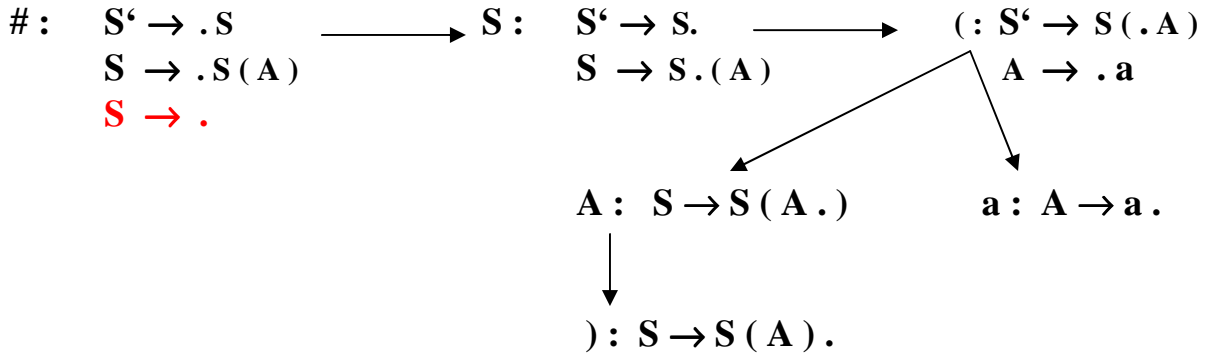
Konfigurace (M_0, w, e) je počáteční a konfigurace (M_0M_i, e, π) je koncová konfigurace algoritmu, kde M_i je symbol na vrcholu zásobníku, při kterém došlo k přijetí ($f(M_i, e) = \text{přijetí}$).

Př. na tabuli pro $G_{12}[E']$

$(\#, a+a*a, -) \vdash (\#a, +a*a, -) \vdash (\#F_1, +a*a, 6) \vdash$
 $\vdash (\#T_1, +a*a, 6\ 4) \vdash (\#E_1, +a*a, 6\ 4\ 2) \vdash \dots$

Př. $G_{13}[S']$: $S' \rightarrow S$ $S \rightarrow S(A) \mid e$ $A \rightarrow a$
 S očíslováním (0) (1) (2) (3)

Množiny položek a graf přechodů mezi nimi mají tvar:



Jak to bude s tím e?

Tabulky mají tvar:

	a	()	e	S	A	a	()
#		2	2	S			
S		P	A				()
A			P)
a			3				
()	P				A	a	
)		1	1				

Proč? Protože jak již víme platí:

$f(\#, u) = \text{přesun}$, jestliže $X \rightarrow \beta_1 \cdot \beta_2 \in \#$, $\beta_2 \in T(N \cup T)^*$ a
 $u \in \text{FIRST}(\beta_2 \text{ FOLLOW}(X))$,
a tento případ zde není

$f(\#, u) = \text{redukce}(j)$, jestliže $X \rightarrow \beta \cdot \in \#$, $X \rightarrow \beta$ je j-té pravidlo v P a
 $u \in \text{FOLLOW}_k(X)$
a tento případ zde je

Zkusme nějakou větu analyzovat

$(\#, (a), -) \vdash (\#S, (a), 2) \vdash (\#S(, a), 2) \vdash (\#S(a,), 2) \vdash$
 $(\#S(A,), 2\ 3) \vdash (\#S(A), e, 2\ 3) \vdash (\#S, e, 2\ 3\ 1)$

LALR(k) gramatiky

V množinách LR(0) položek může nastat konflikt redukce-redukce nebo redukce-přesun, který lze odstranit zohledněním dopředu prohlížených symbolů v obecnějších tzv. LALR(k) položkách („look ahead LR(k)“)

Definice

$LR(k)$ položka pro bezkontextovou gramatiku

$G = (N, T, P, S)$

je objekt tvaru:

$[A \rightarrow \alpha \cdot \beta, w]$

 to, co je za β , nebo-li
co může být právě za tímto A , nebo-li
 $w \in$ podmnožiny FOLLOW(A)

kde $A \rightarrow \alpha \beta \in P$ a $w \in T^*$, $|w| \leq k$ je tzv. dopředu prohlížený řetěz terminálních symbolů délky nejvýše k .

Dále nadefinujeme pojem jádra položek v množině $LR(k)$ položek.

Definice

Nechť M je množina $LR(k)$ položek. Jádro J množiny položek M je množina:

$$J(M) = \{A \rightarrow \alpha \cdot \beta : [A \rightarrow \alpha \cdot \beta, w] \in M\}.$$

Nyní můžeme uvést algoritmus pro výpočet souboru množin položek pro LALR(k) gramatiku, tj. ; soubor množin LALR(k) položek.

Algoritmus

Výpočet souboru množin $LALR(k)$ položek pro $G = (N, T, P, S)$

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

Výstup: Soubor φ množin $LALR(k)$ položek pro G .

Metoda:

1. Počáteční množinu $LALR(k)$ položek M_0 vytvoříme takto:

(a) $M_0 = \{[S \rightarrow \cdot \omega, e] : S \rightarrow \omega \in P\}$.

(b) Jestliže $[A \rightarrow \cdot B \alpha, u] \in M_0, B \in N$ a $B \rightarrow \beta \in P$, pak

$M_0 = M_0 \cup \{[B \rightarrow \cdot \beta, x] : \text{pro všechna } x \in \text{FIRST}_k(\alpha u)\}$.

(c) Opakujeme krok (b) tak dlouho, dokud je možno do M_0 přidávat nové položky.

(d) $\varphi = \{M_0\}$, M_0 je počáteční množina.

2. Jestliže jsme zkonstruovali množinu $LALR(k)$ položek M_i ,

zkonstruujeme pro každý symbol $X \in (N \cup T)$ takový, že leží v některé $LALR(k)$ položce v M_i za tečkou další množinu $LALR(k)$

položek M_j , kde j je větší než nejvyšší index dosud vytvořené množiny $LALR(k)$ položek v φ , takto:

(a) $M_j = \{[A \rightarrow \alpha X \cdot \beta, u] : [A \rightarrow \alpha \cdot X \beta, u] \in M_i\}$.

(b) Jestliže $[A \rightarrow \alpha \cdot B \beta, u] \in M_j, B \in N, B \rightarrow \gamma \in P$, pak

$M_j = M_j \cup \{[B \rightarrow \cdot \gamma, x] : \text{pro všechna } x \in \text{FIRST}_k(\beta u)\}$.

(c) Opakujeme krok (b) tak dlouho, dokud je možné do M_j přidávat nové položky.

(d) Jestliže jádro $J(M_j) \neq J(M_n)$ pro všechna $M_n \in \varphi$, pak $\varphi = \varphi \cup \{M_j\}$

a $\text{GOTO}(M_i, X) = M_j$.

Jestliže $J(M_j) = J(M_n)$ pro nějaké $M_n \in \varphi$, pak

$M_n' = M_n \cup M_j$ a $\varphi = (\varphi - \{M_n\}) \cup \{M_n'\}$ a $\text{GOTO}(M_i, X) = M_n'$.

3. Krok 2. opakujeme pro všechny vytvořené množiny v φ tak dlouho, dokud je možné vytvářet nové množiny M_j .

Definice

Bezkontextovou gramatiku $G = (N, T, P, S)$ nazveme $LALR(k)$ gramatikou ($k \geq 0$) právě tehdy, když v souboru množin $LALR(k)$ položek vytvořených podle algoritmu pro rozšířenou gramatiku G' nejsou žádné konflikty.

Poznámka:

V dalších případech budeme pro položky

$$[A \rightarrow \alpha \cdot \beta, x_1], [A \rightarrow \alpha \cdot \beta, x_2], \dots [A \rightarrow \alpha \cdot \beta, x_n]$$

používat zkráceného zápisu $[A \rightarrow \alpha \cdot \beta, x_1|x_2| \dots |x_n]$.

Jestliže v množině $LALR(k)$ položek nejsou žádné konflikty, můžeme sestrojit tabulku akcí f pro syntaktický analyzátor pomocí následujícího algoritmu.

Tabulka přechodů se vytvoří stejně jako pro $SLR(k)$ gramatiku na základě funkce GOTO.

Algoritmus sestrojení tabulky akcí pro $LALR(k)$ gramatiku.

Vstup: Soubor množin φ $LALR(k)$ položek pro gramatiku $G = (N, T, P, S)$.

Výstup: Tabulka akcí f pro gramatiku G .

Metoda: Tabulka akcí f bude mít řádky označeny stejně jako množiny z φ .

Sloupce budou označeny řetězy symbolů $u \in T^*$, $|u| \leq k$.

Pro všechna $i \in \langle 0, |\varphi| \rangle$ provedeme:

- a) $f(M_i, u) = \text{přesun}$, jestliže $[A \rightarrow \beta_1 \cdot \beta_2, v] \in M_i$, $\beta_2 \in T(N \cup T)^*$ a $u \in \text{FIRST}_k(\beta_2 v)$.
- b) $f(M_i, u) = \text{redukce}(j)$, jestliže $[A \rightarrow \beta \cdot, u] \in M_i$ a $A \rightarrow \beta$ je j -té pravidlo v P , kromě situace podle c),
- c) $f(M_i, e) = \text{přijetí}$, jestliže $[S' \rightarrow S \cdot, e] \in M_i$ a vstupní řetěz je přečten,
- d) $f(M_i, n) = \text{chyba}$ v ostatních případech.

Př. G_{14} na tabuli ? (stránka)

$S \rightarrow L = R$

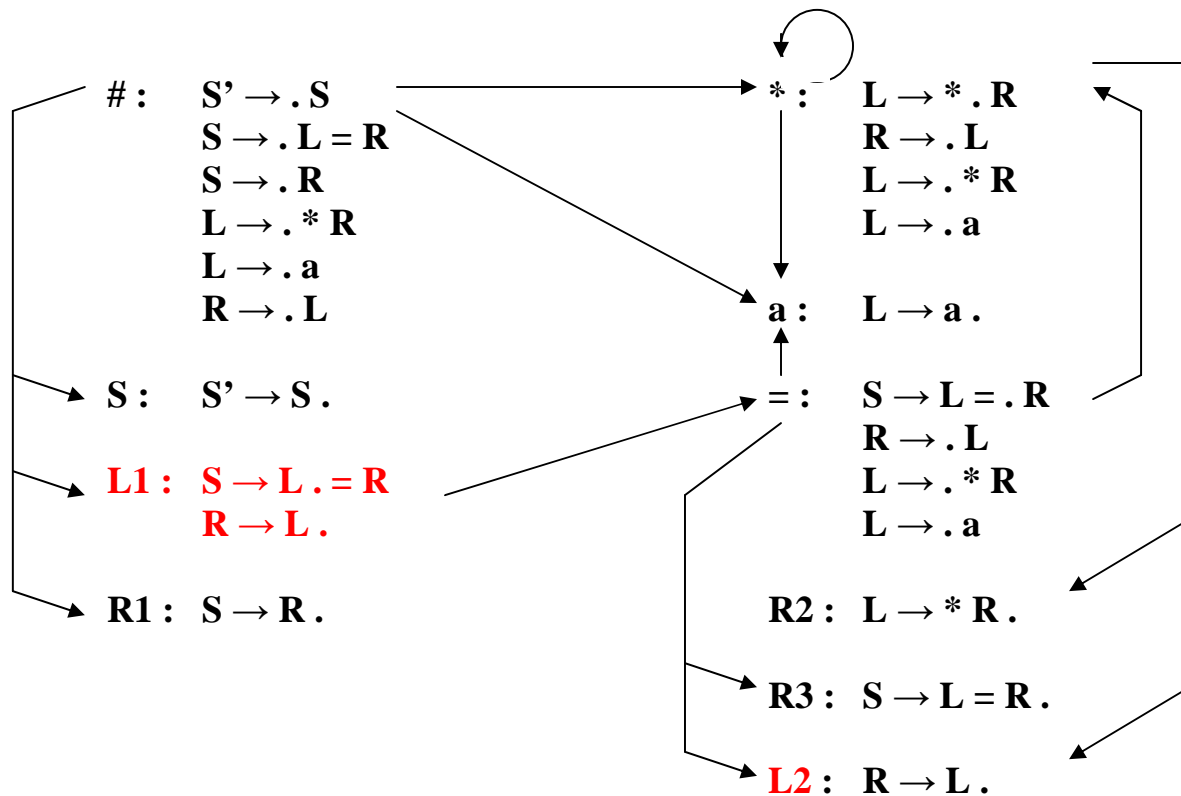
$S \rightarrow R$

$L \rightarrow * R$

$L \rightarrow a$

$R \rightarrow L$

Zkonstruuje soubor množin $LR(0)$ položek pro rozšířenou gramatiku $G_{14}[S']$

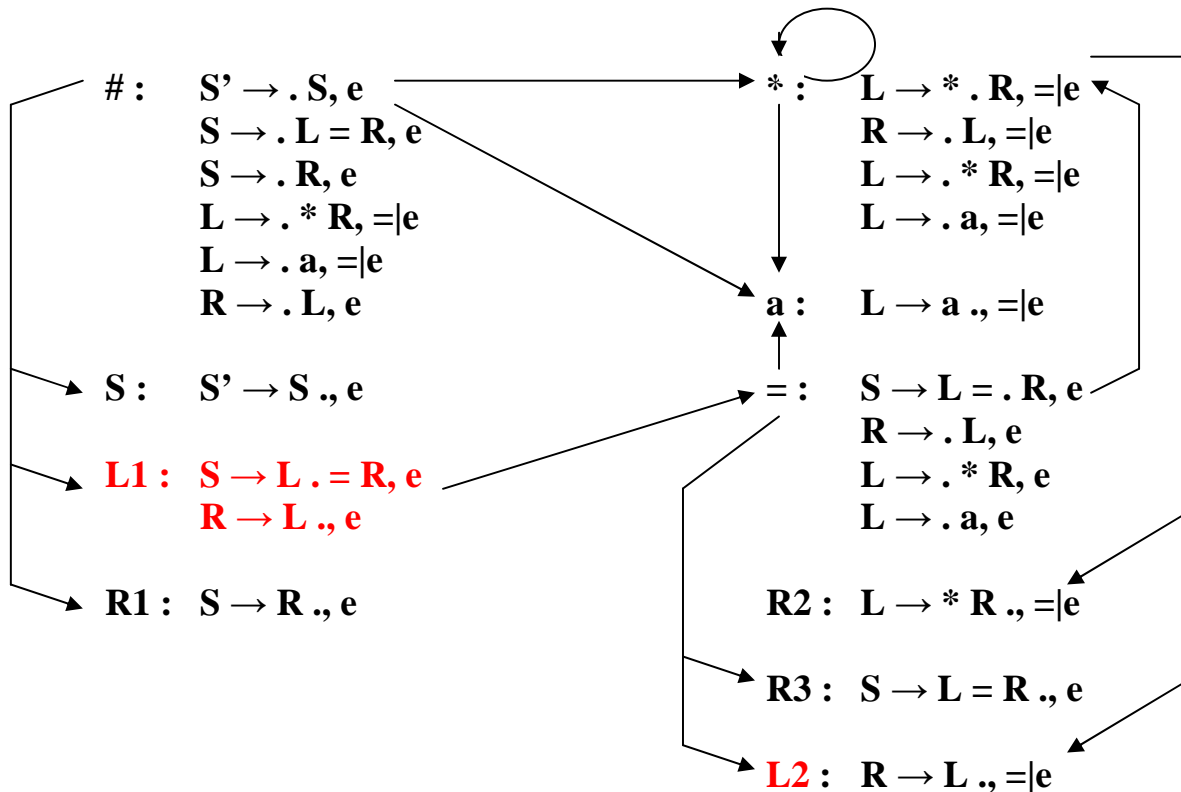


A máme tady problém v L1, která zřetelně indikuje pro „=” přesun. Pro symboly z FOLLOW(R) by se mělo redukovat podle pravidla $R \rightarrow L$. Do množiny FOLLOW(R) ale patří i „=“ . G_{14} není proto SLR(1). Podíváme se tedy co může následovat za pravou stranou (look ahead) a zkonstruujeme množiny LALR(1) položek.

Rozšířená gramatika $G_{14}[S']$

- 0 $S' \rightarrow S$
- 1 $S \rightarrow L = R$
- 2 $S \rightarrow R$
- 3 $L \rightarrow * R$
- 4 $L \rightarrow a$
- 5 $R \rightarrow L$

Množiny LALR(1) položek



	a	=	*	e	a	=	*	S	L	R
#	P		P		a		*	S	L1	R1
S				A						
L1		P		5		=				
R1				2						
R2		3		3						
R3				1						
*	P		P		a		*		L2	R2
a		4		4						
=	P		P		a		*		L2	R3
L2		5		5						

Ted' zkuste analýzu nějakého řetězce

LR(k) gramatiky

Pro výpočet množin LR(k) položek je třeba jen v algoritmu pro výpočet souboru množin *LALR(k)* položek změnit bod 2d) takto:

$$2d) \varphi = \varphi \cup \{M_j\}, \text{GOTO}(M_i, X) = M_j.$$

To znamená, že vytvořená množina položek M_j se přidá do souboru φ i tehdy, když v φ je již množina položek se stejným jádrem, ale jinými dopředu prohlíženými symboly.

Jestliže algoritmus upravíme tak, že změníme bod 2d), dostaneme algoritmus pro výpočet souboru množin *LR(k)* položek.

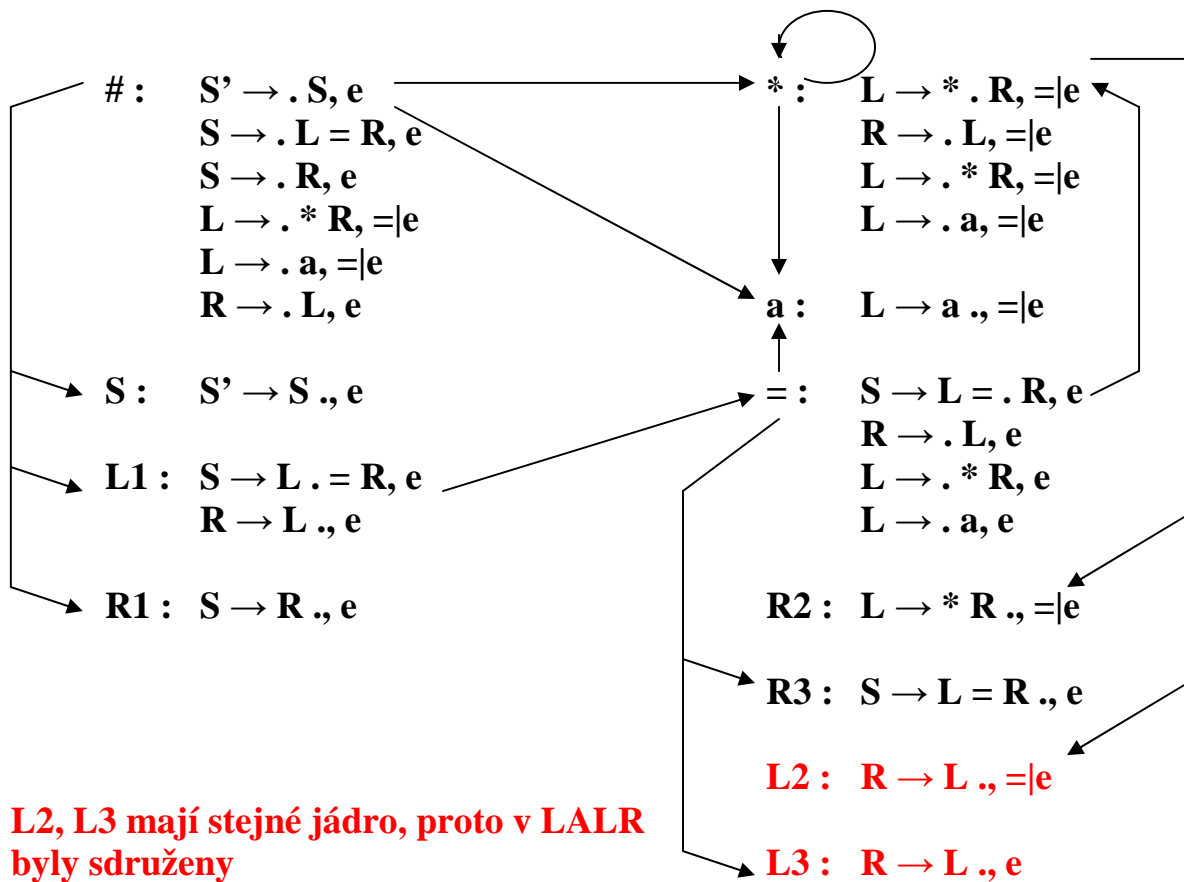
Pro vytvoření tabulky akcí a tabulky přechodů na základě souboru *LR(k)* položek můžeme pak použít tytéž algoritmy jako pro *LALR(k)* gramatiky.

Definice:

Bezkontextovou gramatiku $G = (N, T, P, S)$ nazveme *LR(k)* gramatikou pro $k \geq 0$ právě tehdy, když v souboru množin *LR(k)* položek vytvořeného podle upraveného algoritmu pro rozšířenou gramatiku G' nejsou žádné konflikty.

Př. $G_{14}[S']$ řešená jako LR(1)

Množiny LR(1) položek



L2, L3 mají stejné jádro, proto v LALR byly sdruženy

LR(1) rozkladová tabulka

	a	=	*	e	a	=	*	S	L	R
#	P		P		a		*	S	L1	R1
S				A						
L1		P		5		=				
R1				2						
R2		3		3						
R3				1						
*	P		P		a		*		L2	R2
a		4		4						
=	P		P		a		*		L3	R3
L2		5		5						
L3				5						

LR $(\#, a = * a, -) \vdash (\# a, = * a, -) \vdash (\# L_1, = * a, 4) \vdash$
LALR dtto

LR $\vdash (\# L_1 =, * a, 4) \vdash (\# L_1 = *, a, 4) \vdash (\# L_1 = * a, e, 4)$
LALR dtto

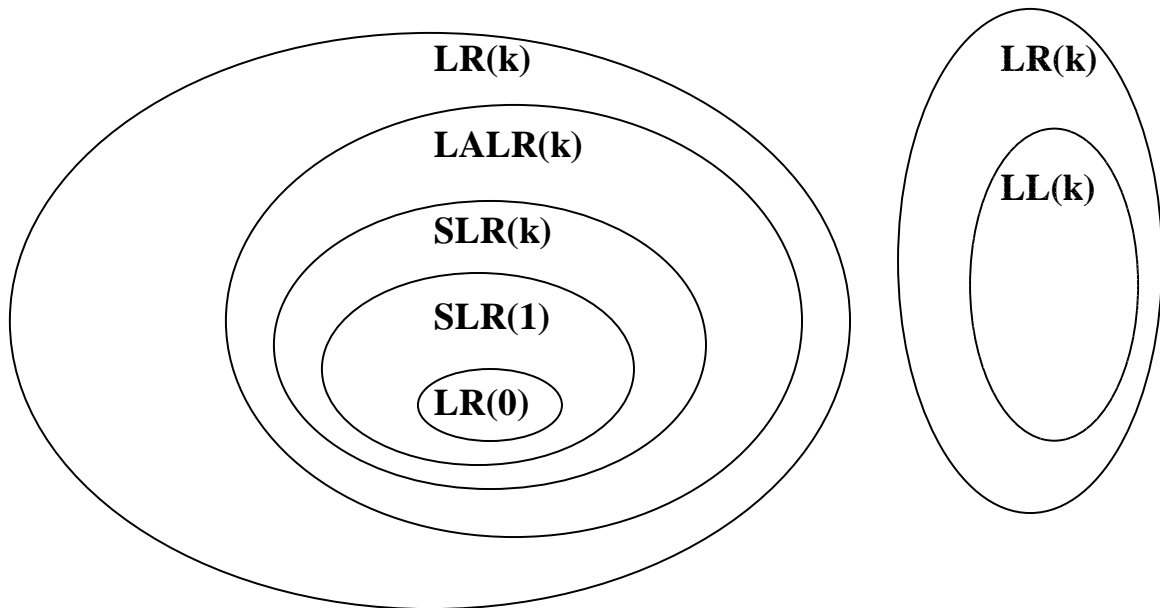
LR $\vdash (\# L_1 = * L_3, e, 44) \vdash (\# L_1 = * R_2, e, 445) \vdash$
LALR $\vdash (\# L_1 = * L_2, e, 44) \vdash$

LR $\vdash (\# L_1 = * R_2, e, 445) \vdash (\# L_1 = * L_3, e, 4453) \vdash$
LALR dtto

LR $\vdash (\# L_1 = R_3, e, 44535) \vdash (\# S, e, 445351)$
LALR dtto

Shrnutí

Platí:



Každá LR(k) i každá LL(k) gramatika je jednoznačná

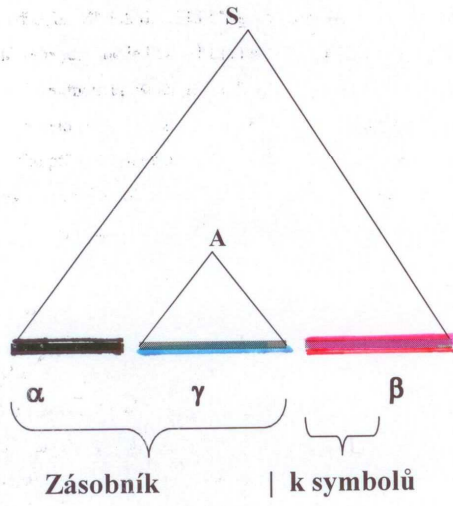
Problém, zda daná gramatika je LR(k) pro zadané k je rozhodnutelný

„ „ „ „ „ „ „ „ „ libovolné k je nerozhodnutelný

Problém, zda pro jazyk \exists LR(k) gramatika je nerozhodnutelný

Každou LR(k) gramatiku lze transformovat redukováním FOLLOW na LR(1).

Proč je LR širší třídou než LL?



2. Kdo má více informace = dohlédne dál²

