LR(k) gramatiky

Př. Uvažujme gramatiku $G_{10}[S]$

1.	$S \rightarrow A\ B$
2.	$A \rightarrow f$
3.	$A \rightarrow a$
4.	$B \rightarrow b\ C$
5.	$C \rightarrow c$

Rozšířený ZA bude mít přechod. fci (viz body z předn. 8 BKG:

dle 1) $\delta(q, i, -) = \{(q, i)\}$ pro $\forall i \in \{a, b, c, f\}$

dle 2) $\delta(q, -, A\ B) = \{(q, S)\}$

$\delta(q, -, f) = \{(q, A)\}$

$\delta(q, -, a) = \{(q, A)\}$

$\delta(q, -, b\ C) = \{(q, B)\}$

$\delta(q, -, c) = \{(q, C)\}$

dle 3) $\delta(q, e, \#\ S) = \{(r, e)\}$

Takový automat je nedeterministický, ! vrchol zásobníku je vpravo !

δ provádí operace: přesouvání , dle 1)
redukování, dle 2)
akceptování. dle 3)

Jeho konfigurací je trojice (stav, vstup, obsah zásobníku)

např. zpracování řetězce f b c (na tabuli)

$(q, fbc, \#) \vdash (q, bc, \#f) \vdash (q, bc, \#A) \vdash (q, c, \#Ab) \vdash \ldots$

Přesun redukce dle 2 přesun

δ je nepřehledné, použijeme tabulky:

tabulka akcí f (co má ZA udělat za operaci)
tabulka přechodů g (co má vložit do zásobníku)

tab.f =

vrchol zásobníku	akce
a	redukce 3
b	přesun
c	redukce 5
f	redukce 2
A	přesun
B	redukce 1
C	redukce 4
S	přijetí
#	přesun

Vkládaný symbol do zásobníku

vrchol zásobníku

tab. g =

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

Tak snadné to je jen u tzv.<u>triviálních</u> 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) = redukce(i),**
b) **f(S) = přijetí**
c) **f(X) = přesun v ostatních případech**

2. Tabulka přechodů g (sloupce jsou $N \cup T$, řádky jsou $N \cup T \cup \{ \# \}$)

α) **g(#, X) = X jestliže v G $\exists$ derivace S $\Rightarrow^*$ X α**
β) **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) = 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) = přesun, přečti vstupní symbol a jdi na 2.**
- **b. Je-li f(X) = 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) = 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) = chyba, ukončíme analýzu chybou.**

Konfiguraci zapisujme 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, -) $\vdash$ (#f, bc, -)
$\vdash$ (#A, bc, 2)
$\vdash$ (#Ab, c, 2)
$\vdash$ (#Abc, e, 2)
$\vdash$ (#AbC, e, 25)
$\vdash$ (#AB, e, 254)
$\vdash$ (#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 → B**	**S → B1**
	2	**B → a B b**	**B → a B2 b1**
	3	**B → A**	**B → A1**
	4	**A → b A**	**A → b2 A2**
	5	**A → c**	**A → 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, -)
├ ($\#ab_2A_2$, b, 5)
├ ($\#aA_1$, b, 5 4)
├ ($\#aB_2$, b, 5 4 3)
├ (#a B_2b_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 =>* X α
Podle dosavadního postupu

proto g(#, S) = S
g(#, B) = B1
g(#, a) = a
g(#, A) = A1
g(#, c) = c
g(#, b) = b2

symboly, které mohou být v zásobníku přímo u #

f(#) = $\boxed{\textbf{přesun}}$

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

 # : S → . B
B → . a B b
B → . A
A → . b A
A → . c

Tečka symbolizuje rozhranní ***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 # mohu 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 φ množin $LR(0)$ položek pro G.

Metoda:

1. **Počáteční množinu $LR(0)$ položek M_0 vytvoříme takto:**
   (a) $M_0 = \{S \to \cdot\, \alpha : S \to \alpha \in P\}$.
   (b) Jestliže $A \to \cdot\, B\alpha \in M_0, B \in N$ a $B \to \beta \in P$, pak
   $M_0 = M_0 \cup \{B \to \cdot \beta\}$.
   (c) Opakujme krok b) tak dlouho, dokud je možné přidávat nové položky do M_0.
   (d) $\varphi = \{M_0\}$, M_0 je počáteční množina.
2. **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 vetší než nejvyšší index dosud vytvořené množiny M_i takto:**

   (a) $M_j = \{A \to \alpha X \cdot \beta : A \to \alpha \cdot X \beta \in M_i\}$.
   (b) Jestliže $A \to \alpha \cdot B \beta \in M_j$, $B \in N$, $B \to \gamma \in P$, pak
   $M_j = M_j \cup \{B \to \cdot\, \gamma\}$.
   (c) Opakujeme krok (b) tak dlouho, dokud je možné do M_j přidávat nové položky.
   (d) $\varphi = \varphi \cup \{M_j\}$.
3. **Krok 2) opakujeme pro všechny vytvořené množiny M_j , dokud je možné do φ 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.

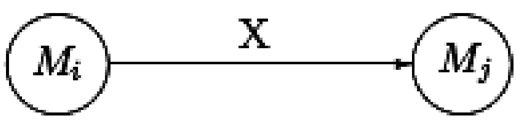
Definice

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

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) $S \to B$
2) $B \to aBb$
3) $B \to A$
4) $A \to bA$
5) $A \to c$

$S' \to .S$

\# :
$S \to .B$
$B \to .aBb$
$B \to .A$
$A \to .bA$
$A \to .c$

$B_1 : S \to B.$

$a : B \to a.Bb$
$B \to .aBb$
$B \to .A$
$A \to .bA$
$A \to .c$

$A_1 : B \to A.$

$b_2 : A \to b.A$
$A \to .bA$
$A \to .c$

$c : A \to c.$

$B_2 : B \to aB.b$

$A_2 : A \to bA.$

$b_1 : B \to aBb.$

$S : S' \to S.$

Když vyplníme podle ~~této~~ modré množiny množin položek tabulky f a g, bude nám chybět údaj do řádku a sloupce pro S. Rozšíříme proto gramatiku o pravidlo 0) $S' \to S$, kde S' bude nový počáteční symbol a doplníme graf

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) = redukce(i), kde i je číslo pravidla $A \rightarrow \alpha$

   Je-li v množině položek X obsažena položka tvaru S' → S.
   pak f(M) = přijetí

   f(M) = 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 GOTO(M_i, X) = M_j, pak g(M_i, X) = M_j**

   b) **Je-li GOTO(M_i, X) = prázdná množina, pak g(M_i, X) = chyba**

Př. Konstruujme 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	$A \rightarrow \alpha\,.$	
tak položku	$B \rightarrow \beta\,.$	**(tzv. konflikt redukce redukce)**
či položku	$C \rightarrow \gamma\,.\,\delta$	**(tzv. konflikt redukce přesun)**

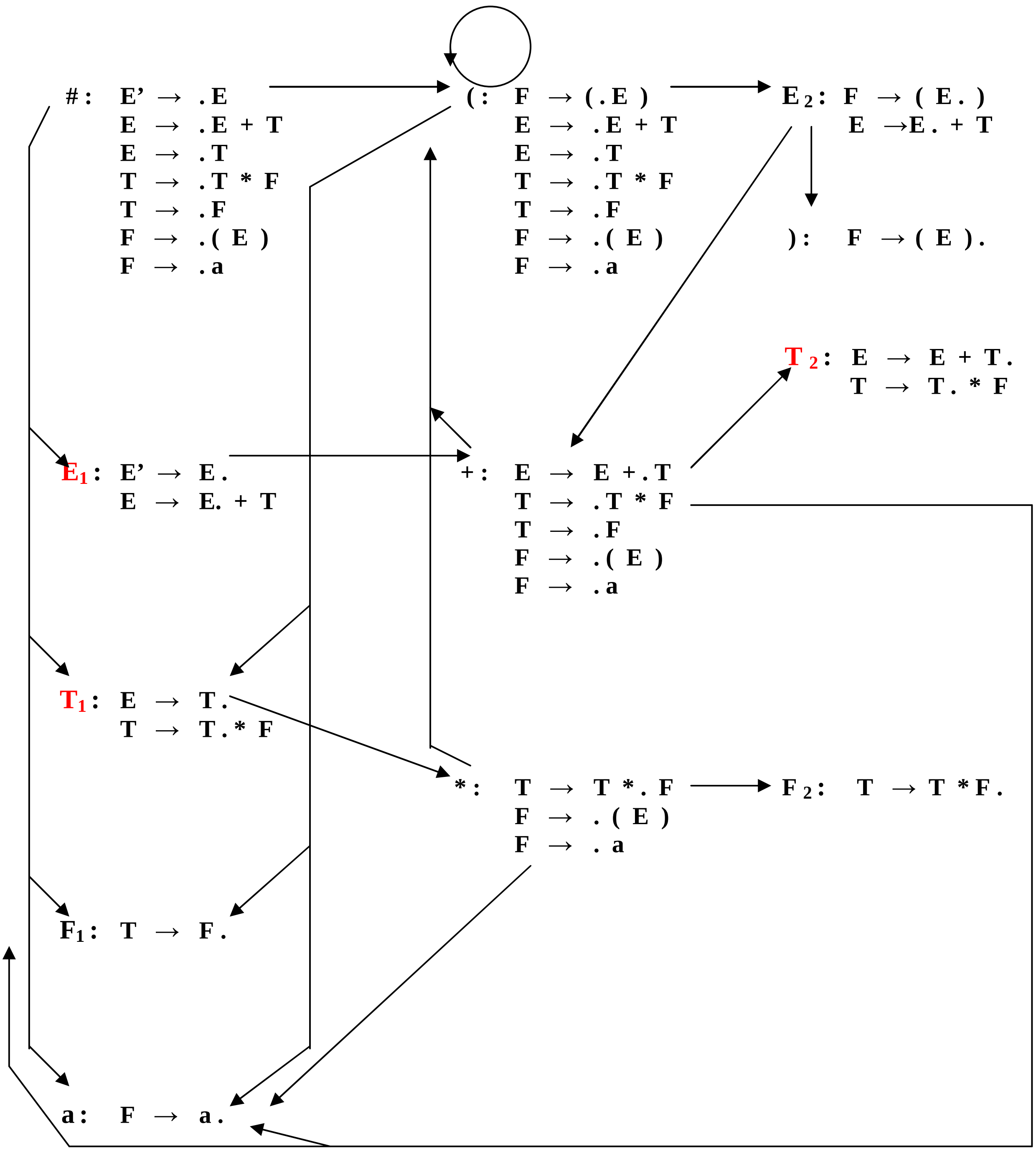
pak gramatika není LR(0). Pokud to půjde napravit prohlížením vstupujících symbolů, pak se jedná o některou z podtříd LR(k) gramatik.

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 $A \to \alpha\,.\,\beta$ a $B \to \gamma\,.\,\delta$ 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 $FOLLOW_k(B) \cap FIRSTk(\beta FOLLOW_k(A)) = \emptyset$ pro $\beta \in T(N \cup T)^*$,**
3. **$\beta = e$, $\delta \neq e$ a $FOLLOW_k(A) \cap FIRST_k(\delta FOLLOW_k(B)) = \emptyset$ pro $\delta \in T(N \cup T)^*$**
4. **$\beta = \delta = e$ a $FOLLOW_k(A) \cap FOLLOW_k(B) = \emptyset$,**

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 sestrojit 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) $f(M_i, u)$ = přesun, jestliže $A \to \beta_1 \cdot \beta_2 \in M_i$, $\beta_2 \in T(N \cup T)^*$ a $u \in FIRST_k(\beta_2\, FOLLOW_k(A))$,
   (b) $f(M_i, u)$ = redukce(j), jestliže $j \geq 1$ a $A \to \beta \cdot \in M_i$, $A \to \beta$ je j-té pravidlo v P a $u \in FOLLOW_k(A)$,
   (c) $f(M_i, e)$ = přijetí, jestliže $S' \to S \cdot \in M_i$,
   (d) $f(M_i, u)$ = chyba ve všech ostatních případech.

2. **Tabulka přechodů g odpovídá funkci GOTO.**
   (a) Je-li GOTO $(M_i, x) = M_j$, kde $x \in (N \cup T)$, pak $g(M_i, x) = M_j$.
   (b) Je-li GOTO (M_i, x) prázdná množina pro $x \in (N \cup T)$, pak $g(M_i, x)$ = chyba.

Př. na tabuli cca 15 řádek

	a	+	*	(	)	e	E	T	F	a	+	*	(	)
#	P			P			E1	T1	F1	a			(	
E1		P				A					+			
T1		2	P		2	2						*		
F1		4	4		4	4								
a		6	6		6	6								
(	P			P			E2	T1	F1	a			(	
+	P			P				T2	F1	a			(	
*	P			P					F2	a			(	
E2		P			P						+			)
T2		1	P		1	1						*		
F2		3	3		3	3								
)		5	5		5	5								

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 $f(X, u)$ = *přesun*, přečte se vstupní symbol a přejdeme na krok 3).**
   (b) Je-li $f(X, u)$ = *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 $f(X, e)$ = *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 $f(X, u)$ = *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 $g(X, Y)$ = *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 ($f(M_i, e)$=*přijetí*).

Př. na tabuli pro G_{12}[E']

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

Př. G_{13}[S']: $S' \to S$ $\quad S \to S\,(\,A\,)\ |\ e$ $\quad A \to a$
S očíslováním (0) (1) (2) (3)
Množiny položek a graf přechodů mezi nimi mají tvar:

#: $S' \to .S$
$\quad S \to .S\,(\,A\,)$
$\quad S \to .$

S: $S' \to S.$
$\quad S \to S.(\,A\,)$

(: $S' \to S\,(\,.A\,)$
$\quad A \to .a$

A: $S \to S\,(\,A\,.)$

a: $A \to a\,.$

): $S \to S\,(\,A\,)\,.$

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:

$f(\#, u)$ = přesun, jestliže $X \to \beta_1 \cdot \beta_2 \in \#$, $\beta_2 \in T(N \cup T)^*$ a
$\quad u \in$ FIRST (β_2 FOLLOW (X)),
a tento případ zde není

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

Zkusme nějakou větu analyzovat
(# , (a) , -) ├ (# S , (a) , 2) ├ (# S (, a) , 2) ├ (# S (a ,) , 2) ├
(# S (A ,) , 2 3) ├ (# S (A) , e , 2 3) ├ (# 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) $M_0 = \{[S \to \cdot\ \omega, e] : S \to \omega \in P\}$.
   (b) Jestliže $[A \to \cdot\ B\alpha, u] \in M_0$, $B \in N$ a $B \to \beta \in P$, pak
   $M_0 = M_0 \cup \{[B \to \cdot\ \beta, x]$: pro všechna $x \in FIRST_k(\alpha u)\}$.
   (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 vetší než nejvyšší index dosud vytvořené množiny $LALR(k)$ položek v φ, takto:
   (a) $M_j = \{[A \to \alpha X \cdot \beta, u] : [A \to \alpha \cdot X\ \beta, u] \in M_i\}$.
   (b) Jestliže $[A \to \alpha \cdot B\ \beta, u] \in M_j$, $B \in N$, $B \to \gamma \in P$, pak
   $M_j = M_j \cup \{[B \to \cdot\ \gamma, x]$: pro všechna $x \in FIRST_k(\beta u)\}$.
   (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 GOTO(M_i,X) = M_j .
   Jestliže $J(M_j) = J(M_n)$ pro nějaké M_n z φ, pak
   $M_n' = Mn \cup M_j$ a $\varphi = (\varphi - \{M_n\}) \cup \{M_n'\}$ a 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

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

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

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*.

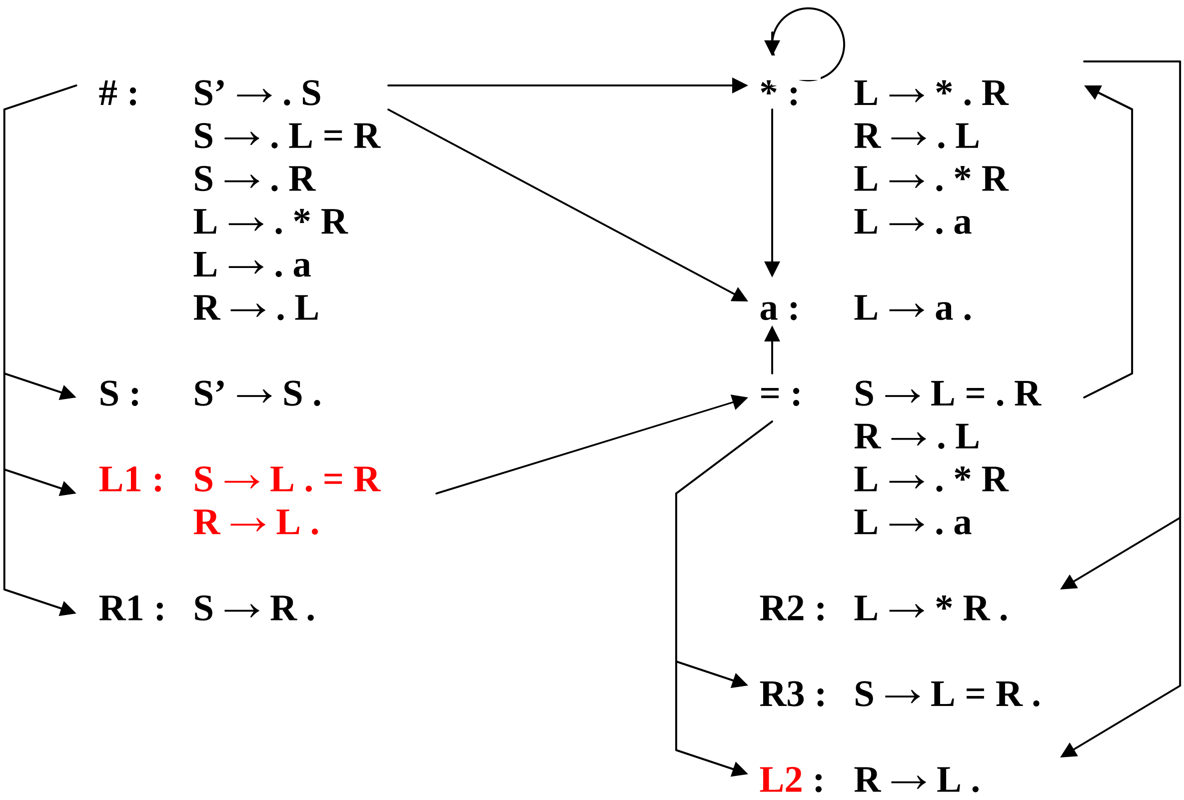
Metoda: Tabulka akcí *f* bude mít řádky označeny stejně jako množiny z φ.
Sloupce budou označeny řetězy symbolů $u \in T^*$, $|u| \le k$.
Pro všechna $i \in \langle 0, |\varphi| \rangle$ provedeme:

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

Př. G_{14} na tabuli ? (stránka)

$S \to L = R$
$S \to R$
$L \to * R$
$L \to a$
$R \to L$

Zkonstruujme 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 → 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’ → S
1	S → L = R
2	S → R
3	L → * R
4	L → a
5	R → L

Množiny LALR(1) položek

 # : S’ → . S, e
S → . L = R, e
S → . R, e
L → . * R, =|e
L → . a, =|e
R → . L, e

S : S’ → S ., e

L1 : S → L . = R, e
R → L ., e

R1 : S → R ., e

*** :** L → * . R, =|e
R → . L, =|e
L → . * R, =|e
L → . a, =|e

a : L → a ., =|e

= : S → L = . R, e
R → . L, e
L → . * R, e
L → . a, e

R2 : L → * R ., =|e

R3 : S → L = R ., e

L2 : R → L ., =|e

	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						

Teď 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:

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

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

 # : S’ → . S, e
S → . L = R, e
S → . R, e
L → . * R, =|e
L → . a, =|e
R → . L, e

S : S’ → S ., e

L1 : S → L . = R, e
R → L ., e

R1 : S → R ., e

*** :** L → * . R, =|e
R → . L, =|e
L → . * R, =|e
L → . a, =|e

a : L → a ., =|e

= : S → L = . R, e
R → . L, e
L → . * R, e
L → . a, e

R2 : L → * R ., =|e

R3 : S → L = R ., e

L2 : R → L ., =|e

L3 : R → L ., e

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

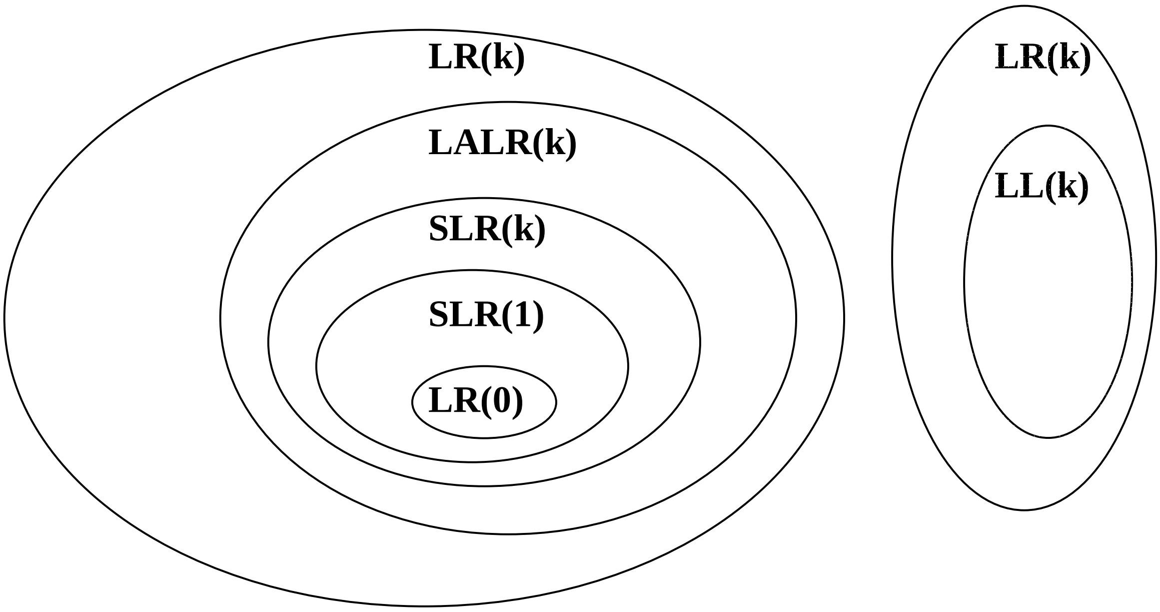
LR(1) rozkladová tabulka

	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	$\vdash (\# L_1 = * L_3 , e , 4 4) \vdash (\# L_1 = * R_2 , e , 4 4 5) \vdash$
LALR	$\vdash (\# L_1 = * L_2 , e , 4 4) \vdash$
LR	$\vdash (\# L_1 = * R_2 , e , 4 4 5) \vdash (\# L_1 = * L_3 , e , 4 4 5 3) \vdash$
LALR	dtto
LR	$\vdash (\# L_1 = R_3 , e , 4 4 5 3 5) \vdash (\# S , e , 4 4 5 3 5 1)$
LALR	dtto

Shrnutí

Platí:



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

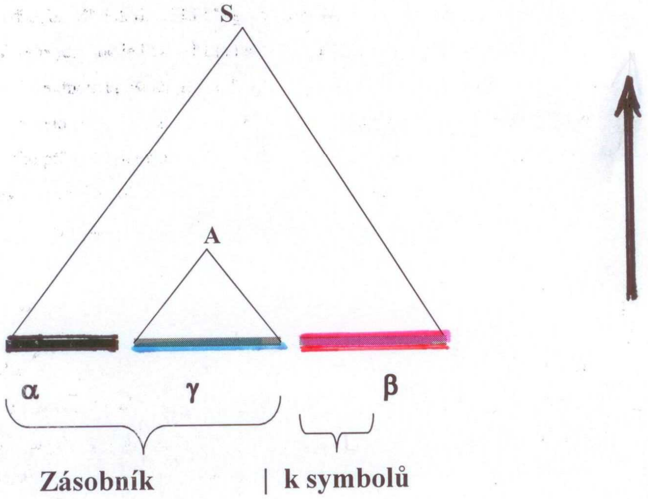
Problém, zda daná gramatika je LR(k) pro zadané k je rozhodnutelný

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

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

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

Proč je LR širší třídou než LL?



Kdo má více informace = dohlédne dál