

Formální překlad

Regulární atributované a překladové gramatiky

Atributovaná gramatika $AG = (G, \text{Atributy}, \text{Sémantická pravidla})$
Atributy jsou přiřazeny symbolům gramatiky a sémantická pravidla jednotlivým prepisovacím pravidlům. Při aplikaci prepisovacího pravidla se provedou příslušná sémantická pravidla a vypočtou hodnoty atributů. Atributy vyhodnocované průchodem derivačním stromem zdola nahoru nazýváme syntetizované, shora dolů nazýváme dědičné.

Překladová gramatika $PG = (N, T \cup D, P, S)$
Obsahuje disjunktí množiny T a D , vstupních a výstupních terminálních symbolů

Regulární pravá překladová gramatika má množinu pravidel tvaru $X \rightarrow a w' Y$, $X \rightarrow a w'$ kde $a \in T$ a $w' \in D^*$,
a nebo $S \rightarrow e$, pokud se S nevyskytuje na pravé straně pravidel.

Př. $PG = (\{S, A, B, C\}, \{i, +, *\} \cup \{i', +', *'\}, P, S)$ s pravidly

$S \rightarrow i i' A$	$S \rightarrow i i'$
$A \rightarrow * C$	$A \rightarrow + B$
$B \rightarrow i i' +' A$	$B \rightarrow i i' +'$
$C \rightarrow i i' *' A$	$C \rightarrow i i' *'$

Derivujme vstupní řetězec $i * i + i$

$S \Rightarrow i i' A \Rightarrow i i' * C \Rightarrow i i' * i i' *' A \Rightarrow i i' * i i' *' + B$
 $\Rightarrow i i' * i i' *' + i i' +'$

Derivací vstupního řetězce vznikl řetěz výstupních symbolů $i' i' *' i'$
 $+'$

Vidíme jej v řetězci $i i' * i i' *' + i i' +'$, „brýlemi výstupního homomorfismu“ (těmi vidíme jen výstupní symboly)

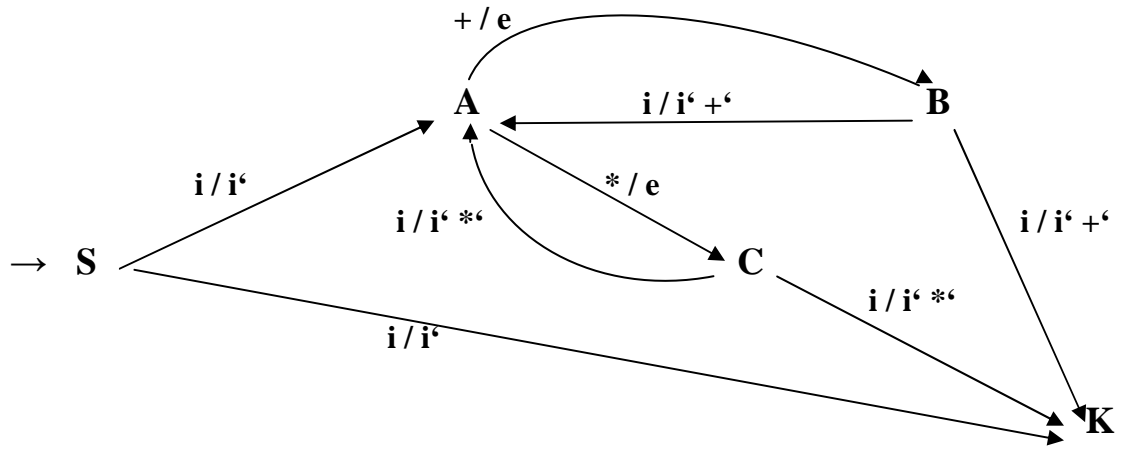
Uvedená gramatika realizuje „nedokonalý“ překlad z infixového zápisu do postfixového. V čem je jeho nedokonalost?

Regulární překladové gramatice odpovídá konečný překladový automat KPA

graf

	A	B	
S			doplňme
	C		
		K	

Výsledek:



Atributovaná překladová gr. APG = (PG, Atributy, Sémantická pravidla)

Př. Popište APG překlad znakového zápisu celých čísel do jeho hodnoty
Gramatika celého čísla

G[C]: $C \rightarrow \check{c} C \mid \check{c}$ je nedeterministické, spravíme to

G[C]: $C \rightarrow \check{c} Z$ je deterministické
 $Z \rightarrow \check{c} Z \mid e$

Překladová gramatika

PG[C]: $T = \{ \check{c} \}$, $D = \{ \text{výstup} \}$
 $C \rightarrow \check{c} Z$
 $Z \rightarrow \check{c} Z \mid e \text{ výstup}$

APG[C]: bude navíc obsahovat atributy symbolů a sémantická pravidla

symbol	atributy	
	dědičné	syntetizované
\check{c}		kód
C	hodnota	
Z	hodnota	
výstup	hodnota	

syntax	sémantická pravidla
$C \rightarrow \check{c} Z$	$Z.hodnota = ord(\check{c}.kód) - ord('0')$
$Z^0 \rightarrow \check{c} Z^1$	$Z^1.hodnota = Z^0.hodnota * 10 + ord(\check{c}.kód) - ord('0')$
$Z \rightarrow e \text{ výstup}$	$výstup.hodnota = Z.hodnota$

Pozn.: Horním indexem odlišujeme stejně pojmenované symboly v pravidle

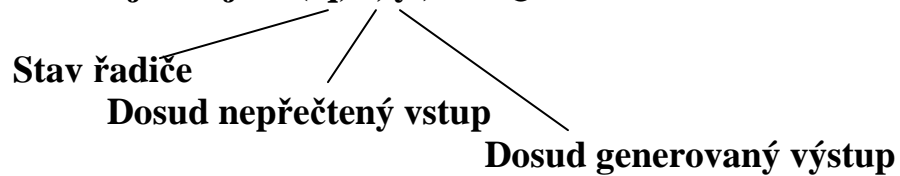
Př. Nakreslete ekvivalentní automat a interpretujte překlad věty 235

Konečný překladačový automat (zopakujme) -KPA je KA rozšířený o výstup

KPA = (Q, T, D, δ , q_0 , F), kde

Q je množina stavů,
 T „ ----- „ vstupních symbolů,
 D „ ----- „ výstupních symbolů,
 $\delta : Q \times (T \cup \{e\}) \rightarrow 2^Q \times D^*$
 q_0 je počáteční stav,
 $F \subset Q$ je množina koncových stavů.

Konfigurace KPA je trojice $(q, x, y) \in Q \times T^* \times D^*$

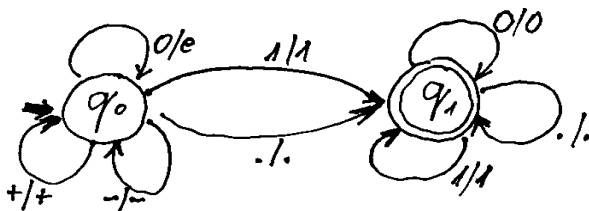


Přechody mezi konfiguracemi označíme $\vdash, \vdash_p, \vdash^*, \vdash_+$

Překlad definovaný KPA = $\{ (u, v) : (q_0, u, e) \vdash^* (r, e, v), r \in F \}$

Př. KPA, který čte binární čísla a vynechává nevýznamové nuly
 (Větve ohodnoceny vstup/výstup)

Pozn. Přeloží ale i jiné řetězce než bin. čísla (vidíte to?)



Zásobníkový překladačný automat ZPA

je ZA rozšířený o výstup resp. KPA s přidáním zásobníkem.

$ZPA = (Q, T, D, \Gamma, \delta, q_0, Z_0, F)$, kde

Q je množina stavů,
 T „ ----- „ vstupních symbolů,
 D „ ----- „ výstupních symbolů,
 Γ „ ----- „ zásobníkových symbolů,
 $\delta : Q \times (T \cup \{e\}) \times \Gamma^* \rightarrow 2^Q \times \Gamma^* \times D^*$
 q_0 je počáteční stav,
 Z_0 je dno zásobníku,
 $F \subset Q$ je množina koncových stavů.

Konfigurace ZPA je $(q, x, y, \alpha) \in Q \times T^* \times D^* \times \Gamma^*$

Stav řadiče

Dosud nepřetčený vstup

dosud generovaný výstup

Obsah zásobníku

Přechod ZA je binární relace $\vdash, \vdash_p, \vdash_*, \vdash_+$ mezi konfiguracemi

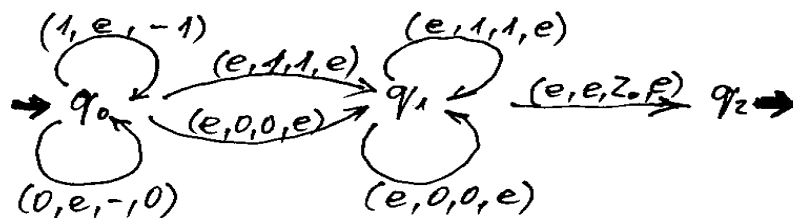
Obdobně jako ZA může i ZPA akceptovat při dočtení vstupního řetězce buď přechodem do koncového stavu nebo vyprázdněním zásobníku.

Př. Ohodnocení větví (*vstup, výstup, starý vrchol, nový vrchol*)

Co to vlastně překládá?

(Vkládá vstupní symbol do zásobníku bez ohledu na vrcholový symbol a po přečtení celého vstupního řetězce vypisuje obsah zásobníku až do jeho vyprázdnění.)

Větve jsou ohodnoceny (vstup, výstup, vrchol, nový vrchol)



Takže překládá řetězce z nul a jedniček na jejich zrcadlový obraz

Překladová bezkontextová gramatika

$PG = (N, T, D, P, S)$ kde

D je množina výstupních terminálních symbolů

Evidentně musí platit $D \cap T = \emptyset$

Pro každou PG existuje ekvivalentní ZPA, tzn. $L(PG) = L(ZPA)$

Konstrukce ekvivalentního ZPA (akceptujícího s prázdným zásob.) k PG

$PG = (N, T, D, P, S)$

$ZPA = (\{q\}, T, D, N \cup T \cup D, \delta, q, S, \emptyset)$

$\delta : \delta(q, -, A) = \{ (q, \alpha, -) : A \rightarrow \alpha \in P \}$ to je expanze

$\delta(q, a, a) = \{ (q, e, e) : \text{pro } \forall a \in T \}$ to je srovnání

$\delta(q, -, b) = \{ (q, e, b) : \text{pro } \forall b \in D \}$ to je výstup
tj. vrchol dá do výstupu

vstup zásobník výstup

Př. Překlad výrazu bez závorek z prefixu do postfixu. Výstupní symboly zde označíme zakroužkováním.

$E \rightarrow +EE(+)$ $E \rightarrow *EE(*)$ $E \rightarrow i(i)$

$(-, b, b, e)$ pro $\forall b \in D$

(a, e, a, e) pro $\forall a \in T$

$(-, e, E, \alpha)$ pro \forall pravé strany α E pravidel

Vstupní G je ale LL(1) \Rightarrow ZPA můžeme být determin.

$(-, e, E, \alpha)$ lze rozpsat na více vřtů

- $(+, e, E, +EE(+))$
- $(*, e, E, *EE(*))$
- $(i, e, E, i(i))$

S významem: (vstup, výstup, vrchol, nový vrchol)

Např. je-li na vstupu „+“ a na vrcholu zásobníku E, nedávej nic do výstupu a vrchol E expanduj na „+ E E ⊕“. Pro „*“ a „i“ je to obdobné.

Atributované překladové bezkontextové gramatiky

APG =

(Překladová gramatika PG, Množina atributů A, Sémantická pravidla SP)

Každému neterminálnímu symbolu $X \in N$ je přiřazena množina dědičných atributů $I(X)$ a množina syntetizovaných atributů $S(X)$. $S(X) \cap I(X) = \emptyset$
Každému vstupnímu symbolu je přiřazena množina syntetizovaných atributů a každému výstupnímu symbolu je přiřazena množina dědičných atributů.

Gramatická pravidla mají tvar $X_0 \rightarrow X_1 X_2 \dots X_n$,

kde $X_0 \in N$ a $X_i \in N \cup T \cup D$ pro $1 \leq i \leq n$

Pro vyhodnocení atributů pomocí SP je nutnou podmínkou aby platilo:

- Hodnoty děd. atributů počátečního symbolu S jsou zadány,
- Hodnoty synt. atributů vstupních terminálních symbolů jsou zadány.
- Pro každý dědičný atribut d symbolu X_i pravé strany pravidla r tvaru $X_0 \rightarrow X_1 X_2 \dots X_m$ je dáno sémantické pravidlo $d = frdi(a_1, a_2, \dots, a_m)$, kde a_1, a_2, \dots, a_m jsou atributy symbolů pravidla r ,
- Pro každý syntetizovaný atribut s symbolu X_0 levé strany pravidla r tvaru: $X_0 \rightarrow X_1 X_2 \dots X_n$, je dáno sémantické pravidlo $s = frs0(a_1, a_2, \dots, a_m)$, kde a_1, a_2, \dots, a_m jsou atributy symbolů pravidla r .

Jsou-li mezi atributy cyklické závislosti, pak je nelze vyhodnotit

Jednoprůchodový překlad je takový, který dovoluje vyhodnotit všechny atributy v průběhu syntaktické analýzy (jeden průchod synt. stromem).

Pro jednoprůchodový překlad je nutné aby platilo, že hodnoty atributů každého symbolu závisí pouze na attributech již zpracovaných symbolů (s vyhodnocenými atributy). Takovou gramatiku zveme L-atributovanou.

Platí pro každé její pravidlo (r): $X_0 \rightarrow X_1 X_2 \dots X_n$

- 1) Pro každý dědičný atribut d symbolu X_i pravé strany $d = frdi(a_1, a_2, \dots, a_m)$, kde a_1, a_2, \dots, a_m jsou buď dědičné atributy symbolu X_0 nebo dědičné a syntetizované atributy symbolů X_1, X_2, \dots, X_{i-1}
- 2) Pro každý syntetizovaný atribut s levostranného symbolu X_0 je $s = frs0(a_1, a_2, \dots, a_m)$, kde a_1, a_2, \dots, a_m jsou buď dědičné atributy symbolu X_0 nebo dědičné a syntetizované atributy symbolů z pravé strany pravidla.

Překlad při LL analýze

Pro LL analýzu musí být splněno:

1. vstupní gramatika (gram. bez výstupních symbolů) splňuje LL podmínky.
2. PG je sémanticky jednoznačná (pravidla se nesmí odlišovat pouze výstupními symboly).
3. je L-atributovaná

Konstrukce rozkladové tabulky je stejná (s uvážením vstupního homomorfismu) jako akceptačního automatu.

Postup zpracování vstupního řetězce je obdobný jako u akceptačního automatu s těmito rozdíly:

1. Do zásobníku jsou ukládány symboly včetně jejich atributů
2. Je-li na vrcholu zásobníku výstupní symbol, je přenesen i s atributy do výstupu
3. Při expanzi se provedou sémantické akce příslušného pravidla

Př. Překlad přiřazovacího příkazu do postfixových instrukcí

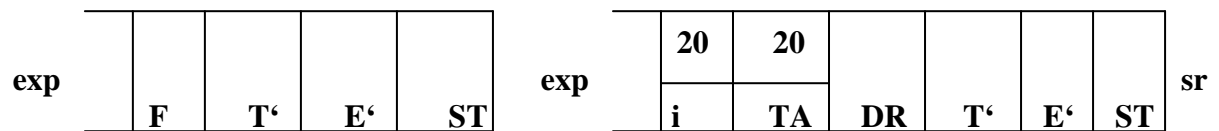
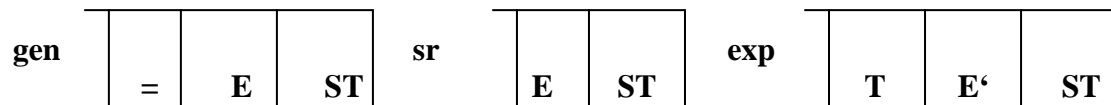
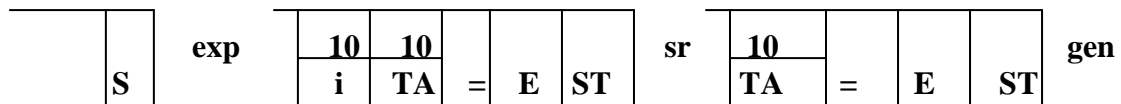
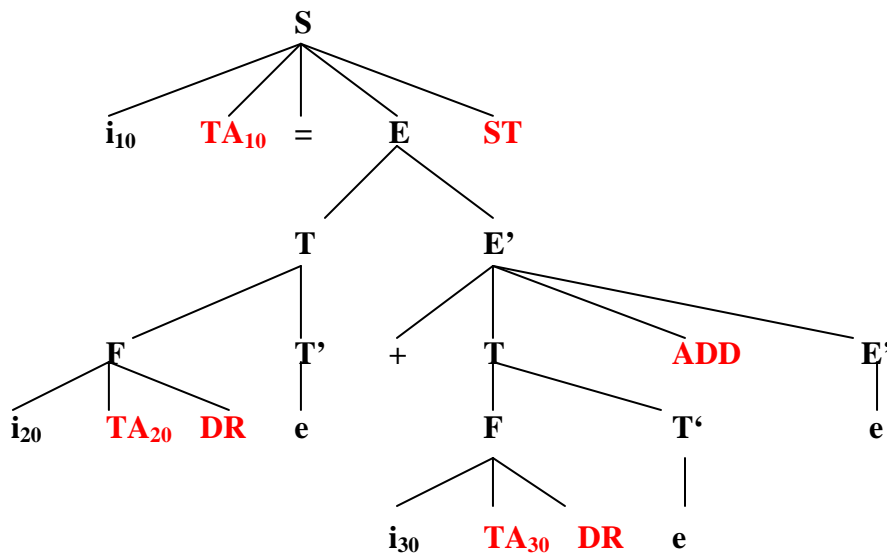
APG= (překl.gr.pro přiřazovací příkaz, s výst. symboly *TA*, *DR*, *ST*, *ADD*, *MUL*
Syntetizovaný atribut *adr*,
Sémantická pravidla)

Gramatická pravidla	sémantická pravidla
1. $S \rightarrow i \quad TA = E \quad ST$	$TA.adr \leftarrow i.adr$
2. $E \rightarrow T \quad E'$	
3. $E' \rightarrow + \quad T \quad ADD \quad E'$	
4. $E' \rightarrow e$	
5. $T \rightarrow F \quad T'$	
6. $T' \rightarrow * \quad F \quad MUL \quad T'$	
7. $T' \rightarrow e$	
8. $F \rightarrow (\quad E \quad)$	
9. $F \rightarrow i \quad TA \quad DR$	$TA.adr \leftarrow i.adr$

Překlad probíhá současně s analýzou:

	=	i	+	*	()	e
S		1					
E		2			2		
E'			3			4	4
T		5			5		
T'			7	6		7	7
F		9			8		

Např. $i_{10} = i_{20} + i_{30}$



sr ...

př.) Překlad přiřazovacího příkazu do čtveřic

Zavedeme pomocnou fci NPP pro generování nové pomocné proměnné, výstupní symboly ASS, ADD, MUL a atributy

Symb.	Atributy	Význam
E	E.ukaz	ukazatel na místo kam dosadit adresu s hodn. E
ASS	ASS.p ASS.l	adresa pravé a levé strany přiřazení
E'	E'.adr E'.ukaz	adresa s hodnotou E', ukazatel kde má být E' použita
ADD	ADD.l ADD.p ADD.v	adresy levého, pravého a výsledkového operandu
MUL	“.....““	“.....““
T	T.ukaz	ukazatel kam dosadit adresu s hodnotou T
T'	T'.adr T'.ukaz	adresa s hodnotou T', ukazatel kde má být T' použita
F	F.ukaz	ukazatel kam dosadit adresu s hodnotou F
i	i.adr	adresa přidělená identifikátoru i

syntaktická pravidla	sémantická pravidla
1) $S \rightarrow i = E \text{ ASS}$	$ASS.l \leftarrow i.adr ; E.ukaz \leftarrow \text{ukazatel na } ASS.p$
2) $E \rightarrow T E'$	$E'.ukaz \leftarrow E.ukaz ; T.ukaz \leftarrow \text{ukazatel na } E'.adr$
3) $E' \rightarrow + T \text{ ADD } E'^1$	$PP \leftarrow NPP ; ADD.v \leftarrow PP ; ADD.l \leftarrow E'.adr ;$ $E'^1.adr \leftarrow PP ; T.ukaz \leftarrow \text{ukazatel na } ADD.p ;$ $E'^1.ukaz \leftarrow E'.ukaz$
4) $E' \rightarrow e$	$\text{kam ukazuje } E'.ukaz \leftarrow E'.adr$
5) $T \rightarrow F T'$	$T'.ukaz \leftarrow T.ukaz ; F.ukaz \leftarrow \text{ukazatel na } T'.adr$
6) $T' \rightarrow * F \text{ MUL } T'^1$	$PP \leftarrow NPP ; MUL.v \leftarrow PP ; MUL.l \leftarrow T'.adr ;$ $T'^1.adr \leftarrow PP ; F.ukaz \leftarrow \text{ukazatel na } MUL.p ;$ $T'^1.ukaz \leftarrow T'.ukaz$
7) $T' \rightarrow e$	$\text{kam ukazuje } T'.ukaz \leftarrow T'.adr$
8) $F \rightarrow (E)$	$E.ukaz = F.ukaz$
9) $F \rightarrow i$	$\text{kam ukazuje } F.ukaz \leftarrow i.adr$

Př. $i_{10} = i_{20} + i_{30}$

S

ex 1

i	10
=	
E	
ASS	10

sr,sr,ex2

T	
E'	
ASS	10

ex 5

F	
T'	
E'	
ASS	10

ex9,sr

T'	20
E'	
ASS	10

ex7

E'	20
ASS	10

ex 3, sr

T		
ADD	20	100
E'	100	
ASS		10

ex5

F		
T'		
ADD	20	100
E'	100	
ASS		10

ex9,sr

T'	30	
ADD	20	100
E'	100	
ASS		10

ex 7

ADD	20	30	100
E'	100		
ASS			10

gen

E'	100		
ASS			10

ex4

ASS	100		10
-----	-----	--	----

gen

Překlad při LR analýze

Př. Překlad přiřazovacího příkazu zdola nahoru do postfixové notace

	=	i	+	*	()	e	E	T	F	LS	+	*	()	=	i	S
#		P									LS						i1	S
E1			P				7					+						
T1			2	P		2	2						*					
F1			4	4		4	4											
i2			6	6		6	6											
(P			P			E2	T1	F1				(i2	
+		P			P				T2	F1				(i2	
*		P			P					F2				(i2	
E2			P			P						+)				
T2			1	P		1	1						*					
F2			3	3		3	3											
)			5	5		5	5											
LS	P															=		
=		P			P			E1	T1	F1				(i2	
S							A											
i1	8																	

Gramatika rozšířená o sémantické akce prováděné při redukcích:

- 0) S' → S
- 1) E → E + T GEN(ADD)
- 2) E → T
- 3) T → T * F GEN(MUL)
- 4) T → F
- 5) F → (E)
- 6) F → i GEN(TA i.adr) ; GEN(DR)
- 7) S → LS = E GEN(ST)
- 8) LS → i GEN(TA i.adr)

Tak jednoduché je to v případech, když APG používá jen syntetizované atributy a výstupní symboly jsou jen na koncích pravých stran pravidel. Taková APG se nazývá S-atributovaná.

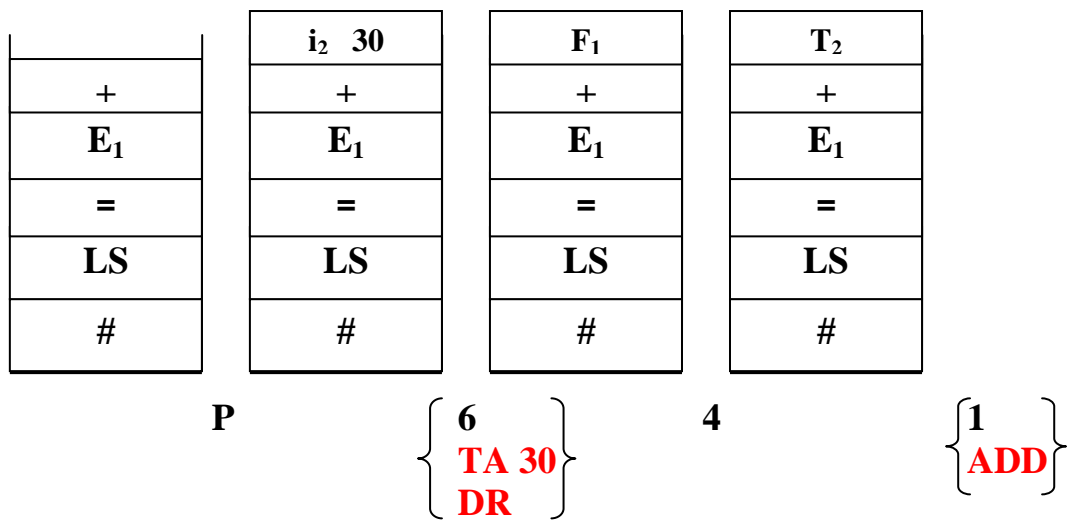
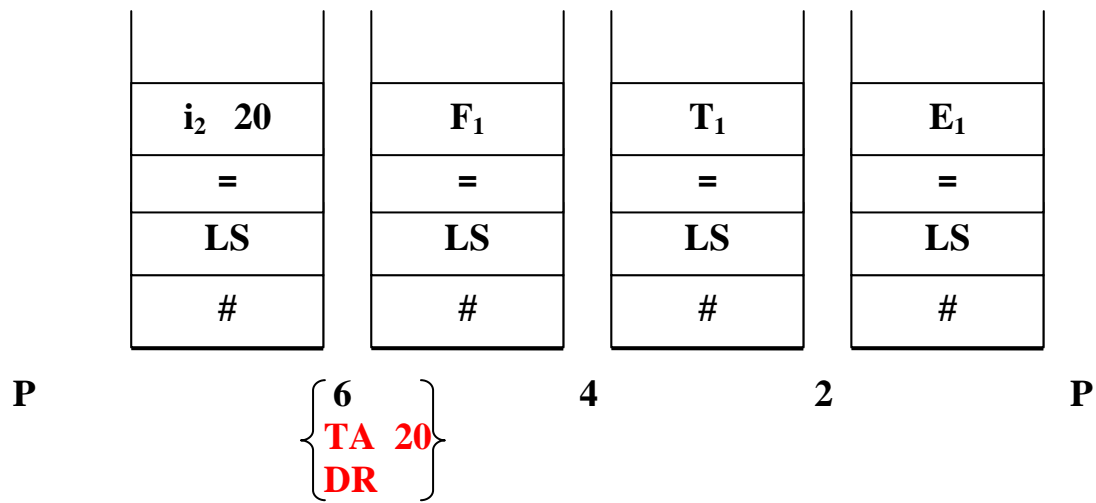
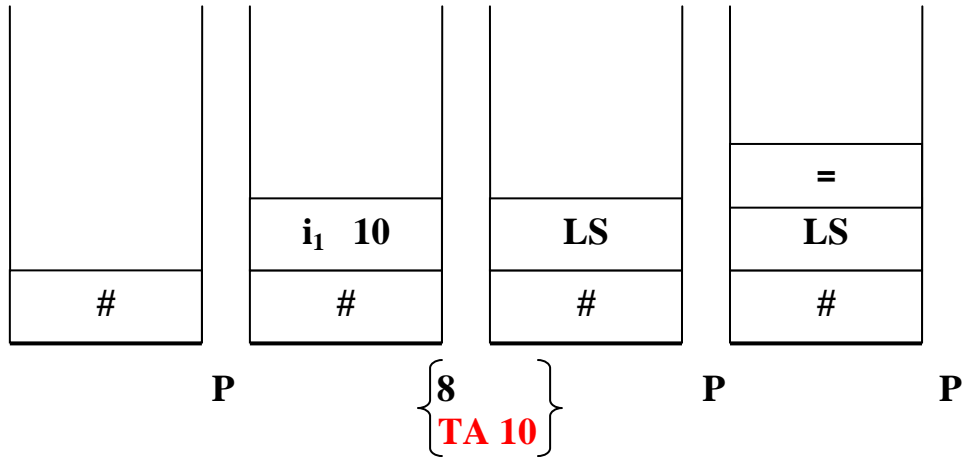
Ve formalismu APG místo akcí generování zavedeme výstupní symboly

syntax				sémantika	
0)	S'	→	S		
1)	E	→	E + T	ADD	
2)	E	→	T		
3)	T	→	T * F	MUL	
4)	T	→	F		
5)	F	→	(E)		
6)	F	→	i	TA	DR
7)	S	→	LS = E	ST	
8)	LS	→	i	TA	

TA .adr = i.adr

TA .adr = i.adr

Př. $i_{10} = i_{20} + i_{30}$

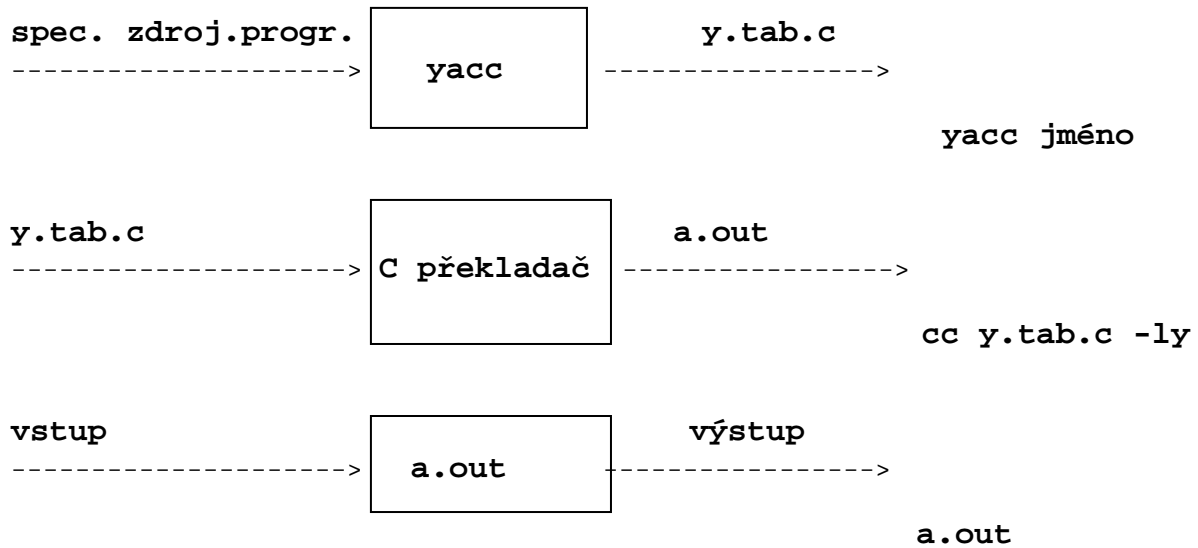


E₁	
=	
LS	S
#	#

7
ST

Akcept

YACC



Tvar zdrojového programu:

```
deklarace
%%
překladová pravidla
%%
pomocné C - programy
```

Tvar překladových pravidel:

```
<levá strana> : <pravá strana 1> {sémantická akce 1}
                | <pravá strana 2> {sémantická akce 2}
                | ...
                | <pravá strana n> {sémantická akce n}
```

Deklarace obsahují:

1. Běžné C deklarace oddělené `{ a }`
2. Deklarace gramatických symbolů, pojmenovávajících terminální symboly

Metasymboly:

```
' ' omezovače terminálního symbolu
$$ hodnota atributu levostranného symbolu
$i hodnota atributu i-tého pravostranného symbolu
```

Pomocné procedury:

Musí vždy uvést lexikální analyzátor `yylex()`.
Lze jej nahradit `#include "lex.yy.c"`
Další pomocné procedury jsou fakultativní

Pracuje s S-atributovanými gramatikami, navíc ale může použít globální proměnné.

Příklad kalkulačky.

Čte aritmetický výraz a vyhodnocuje jej

```
%{
#include <ctype.h>
%}
%token DIGIT
%%
line      :      expr '\n'          {printf("%d\n", $1);}
          ;
expr      :      expr '+' term      {$$ = $1 + $3; }
          |      term
          ;
term      :      term '*' factor    {$$ = $1 * $3; }
          |      factor
          ;
factor    :      '(' expr ')'       {$$ = $2; }
          |      DIGIT
          ;
%%
yylex() {
    int c;
    c = getchar();
    if (isdigit(c)) {
        yylval = c-'0';
        return DIGIT;
    }
    return c;
}
```

Voláním `yacc -v jmenosouboru`
vytvoří navíc soubor `y.out`

```
state 0
  $accept : _line $end

  DIGIT shift 6
  ( shift 5
  . error

  line goto 1
  expr goto 2
  term goto 3
  factor goto 4

state 1
  $accept : line_$end

  $end accept
  . error

state 2
  line : expr_\n
  expr : expr_+ term

  \n shift 7
  + shift 8
  . error

state 3
  expr : term_ (3)
  term : term_* factor

  * shift 9
  . reduce 3

state 4
  term : factor_ (5)

  . reduce 5
```

```

state 5
  factor : ( _expr )

  DIGIT shift 6
  ( shift 5
  . error

  expr goto 10
  term goto 3
  factor goto 4

state 6
  factor : DIGIT_      (7)

  . reduce 7

state 7
  line : expr \n_      (1)

  . reduce 1

state 8
  expr : expr +_term

  DIGIT shift 6
  ( shift 5
  . error

  term goto 11
  factor goto 4

state 9
  term : term *_factor

  DIGIT shift 6
  ( shift 5
  . error

  factor goto 12

state 10
  expr : expr_+ term
  factor : ( expr_)

  + shift 8
  ) shift 13
  . error

```

```
state 11
  expr :  expr + term_    (2)
  term :  term_ * factor
```

```
* shift 9
. reduce 2
```

```
state 12
  term :  term * factor_  (4)
```

```
. reduce 4
```

```
state 13
  factor : ( expr )_    (6)
```

```
. reduce 6
```

```
8/200 terminals, 4/300 nonterminals
8/600 grammar rules, 14/750 states
0 shift/reduce, 0 reduce/reduce conflicts reported
8/350 working sets used
memory: states,etc. 72/12000, parser 9/12000
9/600 distinct lookahead sets
4 extra closures
14 shift entries, 1 exceptions
7 goto entries
3 entries saved by goto default
Optimizer space used: input 40/12000, output 218/12000
218 table entries, 204 zero
maximum spread: 257, maximum offset: 42
```

Příklad kalkulačky pro reálná čísla
(používá nejednoznačnou gramatiku)

```
%{
#include <ctype.h>
#include <stdio.h>
#define YYSTYPE double /*double typ pro Yacc stack*/
%}
%token NUMBER
%left '+' '-'
%left '*' '/'
%right UMINUS

%%
lines : lines expr '\n' {printf("%g\n", $2); }
      | lines '\n'
      /* e */
;
expr  : expr '+' expr      { $$ = $1 + $3; }
      | expr '-' expr     { $$ = $1 - $3; }
      | expr '*' expr     { $$ = $1 * $3; }
      | expr '/' expr     { $$ = $1 / $3; }
      | '(' expr ')'      { $$ = $2; }
      | '-' expr %prec UMINUS { $$ = - $2; }
      | NUMBER
;

%%
yylex() {
    int c;
    while ( ( c = getchar() ) == ' ');
    if ( ( c == '.' ) || ( isdigit(c) ) ) {
        ungetc(c, stdin);
        scanf("%lf", &yylval);
        return NUMBER;
    }
    return c;
}
}
```

LALR algoritmus s konflikty řeší YACC takto:

Konflikt redukce-redukce řeší výběrem pravidla dle pořadí jejich uvedení.

Konflikt redukce-přesun řeší upřednostněním přesunu.