Metody Realizacji Języków Programowania

Analiza składniowa wstępująca

Marcin Benke

MIM UW

25 października 2010

Analiza wstępująca — metoda LR

- Od Lewej, pRawostronne wyprowadzenie (w odwrotnej kolejności)
- Automat ze stosem, na stosie ciąg terminali i nieterminali
- Jeśli na stosie jest prawa strona produkcji, możemy ją zastąpić symbolem z lewej (redukcja)
- Pytanie, kiedy to robić — poznamy różne techniki.
- Automat startuje z pustym stosem i akceptuje, gdy całe wejście zredukuje do symbolu startowego.

Przykład

Gramatyka: $1 : E \to E + T$ $2 : E \to T$,
$3 : T \to T * F$, $4 : T \to F$, $5 : F \to n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)

Przykład

Gramatyka: $1 : E \to E + T$ $2 : E \to T$,
$3 : T \to T * F$, $4 : T \to F$, $5 : F \to n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5:$F \to n$)

Przykład

Gramatyka: $1 : E \to E + T$ $2 : E \to T$,
$3 : T \to T * F$, $4 : T \to F$, $5 : F \to n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5:$F \to n$)
F	+ 2 * 3	reduce 4:$T \to F$

Przykład

Gramatyka: $1 : E \to E + T$ $2 : E \to T$,
$3 : T \to T * F$, $4 : T \to F$, $5 : F \to n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5:$F \to n$)
F	+ 2 * 3	reduce 4:$T \to F$
T	+ 2 * 3	reduce 2:$E \to T$

Przykład

Gramatyka: $1: E \to E + T$ $2: E \to T$,
$3: T \to T * F$, $4: T \to F$, $5: F \to n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5:$F \to n$)
F	+ 2 * 3	reduce 4:$T \to F$
T	+ 2 * 3	reduce 2:$E \to T$
E	+ 2 * 3	shift

Przykład

Gramatyka: $1 : E \to E + T$ $2 : E \to T$,
$3 : T \to T * F$, $4 : T \to F$, $5 : F \to n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5:$F \to n$)
F	+ 2 * 3	reduce 4:$T \to F$
T	+ 2 * 3	reduce 2:$E \to T$
E	+ 2 * 3	shift
E +	2 * 3	shift

Przykład

Gramatyka: $1 : E \to E + T$ $2 : E \to T$,
$3 : T \to T * F$, $4 : T \to F$, $5 : F \to n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5:$F \to n$)
F	+ 2 * 3	reduce 4:$T \to F$
T	+ 2 * 3	reduce 2:$E \to T$
E	+ 2 * 3	shift
E +	2 * 3	shift
E + 2	* 3	reduce 5:$F \to n$

Przykład

Gramatyka: $1 : E \to E + T$ $2 : E \to T$,
$3 : T \to T * F$, $4 : T \to F$, $5 : F \to n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5:$F \to n$)
F	+ 2 * 3	reduce 4:$T \to F$
T	+ 2 * 3	reduce 2:$E \to T$
E	+ 2 * 3	shift
E +	2 * 3	shift
E + 2	* 3	reduce 5:$F \to n$
E + F	* 3	reduce 4:$T \to F$

Przykład

Gramatyka: $1 : E \to E + T$ $2 : E \to T$,
$3 : T \to T * F$, $4 : T \to F$, $5 : F \to n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5:$F \to n$)
F	+ 2 * 3	reduce 4:$T \to F$
T	+ 2 * 3	reduce 2:$E \to T$
E	+ 2 * 3	shift
E +	2 * 3	shift
E + 2	* 3	reduce 5:$F \to n$
E + F	* 3	reduce 4:$T \to F$
E + T	* 3	shift (dlaczego?)

Przykład

Gramatyka: $1 : E \to E + T$ $2 : E \to T$,
$3 : T \to T * F$, $4 : T \to F$, $5 : F \to n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5:$F \to n$)
F	+ 2 * 3	reduce 4:$T \to F$
T	+ 2 * 3	reduce 2:$E \to T$
E	+ 2 * 3	shift
E +	2 * 3	shift
E + 2	* 3	reduce 5:$F \to n$
E + F	* 3	reduce 4:$T \to F$
E + T	* 3	shift (dlaczego?)
E + T *	3	shift

Przykład

Gramatyka: $1 : E \to E + T$ $2 : E \to T$,
$3 : T \to T * F$, $4 : T \to F$, $5 : F \to n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5:$F \to n$)
F	+ 2 * 3	reduce 4:$T \to F$
T	+ 2 * 3	reduce 2:$E \to T$
E	+ 2 * 3	shift
E +	2 * 3	shift
E + 2	* 3	reduce 5:$F \to n$
E + F	* 3	reduce 4:$T \to F$
E + T	* 3	shift (dlaczego?)
E + T *	3	shift
E + T * 3	#	reduce 5:$F \to n$

Przykład

Gramatyka: $1 : E \to E + T$ $2 : E \to T$,
$3 : T \to T * F$, $4 : T \to F$, $5 : F \to n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5:$F \to n$)
F	+ 2 * 3	reduce 4:$T \to F$
T	+ 2 * 3	reduce 2:$E \to T$
E	+ 2 * 3	shift
E +	2 * 3	shift
E + 2	* 3	reduce 5:$F \to n$
E + F	* 3	reduce 4:$T \to F$
E + T	* 3	shift (dlaczego?)
E + T *	3	shift
E + T * 3	#	reduce 5:$F \to n$
E + T * F	#	reduce 3:$T \to T * F$

Przykład

Gramatyka: $1 : E \to E + T$ $2 : E \to T$,
$3 : T \to T * F$, $4 : T \to F$, $5 : F \to n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5:$F \to n$)
F	+ 2 * 3	reduce 4:$T \to F$
T	+ 2 * 3	reduce 2:$E \to T$
E	+ 2 * 3	shift
E +	2 * 3	shift
E + 2	* 3	reduce 5:$F \to n$
E + F	* 3	reduce 4:$T \to F$
E + T	* 3	shift (dlaczego?)
E + T *	3	shift
E + T * 3	#	reduce 5:$F \to n$
E + T * F	#	reduce 3:$T \to T * F$
E + T	#	reduce 1

Przykład

Gramatyka: $1 : E \to E + T$ $2 : E \to T$,
$3 : T \to T * F$, $4 : T \to F$, $5 : F \to n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5:$F \to n$)
F	+ 2 * 3	reduce 4:$T \to F$
T	+ 2 * 3	reduce 2:$E \to T$
E	+ 2 * 3	shift
E +	2 * 3	shift
E + 2	* 3	reduce 5:$F \to n$
E + F	* 3	reduce 4:$T \to F$
E + T	* 3	shift (dlaczego?)
E + T *	3	shift
E + T * 3	#	reduce 5:$F \to n$
E + T * F	#	reduce 3:$T \to T * F$
E + T	#	reduce 1
E	#	accept

Problemy

1. Czy na szczycie stosu jest prawa strona jakiejś produkcji? (łatwe, ale może być więcej niż jedna)
2. Czy należy redukować, a jeśli tak, to którą produkcję?

Tworzymy deterministyczny automat skończony, symulujący prawostronne wyprowadzenie. Automat wykrywa uchwyty (produkcje wraz z miejscem wystąpienia).

Definicja (uchwyt)

$A \to \beta$ jest *uchwytem* w prawostronnej formie zdaniowej $\alpha\beta w$, jeśli

$$S \to^R \alpha A w \to \alpha\beta w$$

dla pewnych $\alpha, \beta \in (N \cup T)^*$, $w \in T^*$.

Gramatyki LR(k)

Nieformalnie: jeśli dla formy zdaniowej αw mamy już na stosie α, to para $\langle \alpha, k : w \rangle$ wyznacza jednoznacznie co zrobić, a w szczególności:

- czy na szczycie stosu jest prawa strona jakiejś produkcji? (łatwe, ale może być więcej niż jedna)
- czy należy redukować, a jeśli tak, to którą produkcję? (trudne, podglądamy k symboli z wejścia)

W praktyce ograniczamy się do $k \leq 1$.

Gramatyki LR(k)

Definicja LR(k)

Gramatyka $G = \langle T, N, S, P \rangle$ jest klasy **LR(k)** jeśli dla dowolnych wyprowadzeń prawostronnych

$$S \to^R \mu A w \to \mu\alpha w$$
$$S \to^R \mu' B w' \to \mu'\beta w'$$

gdzie $\mu, \mu' \in (N \cup T)^*$, $A \to \alpha$, $B \to \beta \in P$ zachodzi

$$\text{jeśli } \mu\alpha \cdot (k : w) = \mu'\beta \cdot (k : w')$$
$$\text{to } \mu = \mu',\ A = B,\ \alpha = \beta$$

Własności gramatyk LR(k)

Twierdzenie (Knuth)

Dla każdej gramatyki LR(k) istnieje równoważny deterministyczny automat ze stosem.

Własności gramatyk LR(k)

Twierdzenie (Knuth)

Dla każdej gramatyki LR(k) istnieje równoważny deterministyczny automat ze stosem.

Twierdzenie (Knuth)

Dla każdego języka deterministycznego L istnieje $k \geq 0$ *i gramatyka* $G \in LR(k)$ *taka, że* $L = L(G)$.

Własności gramatyk LR(k)

Twierdzenie (Knuth)

Dla każdej gramatyki LR(k) istnieje równoważny deterministyczny automat ze stosem.

Twierdzenie (Knuth)

Dla każdego języka deterministycznego L istnieje $k \geq 0$ *i gramatyka* $G \in LR(k)$ *taka, że* $L = L(G)$.

Twierdzenie

Każda gramatyka LL(k) jest też LR(k). Istnieją gramatyki LR(k), które nie są LL(n) dla żadnego n.

Własności gramatyk LR(k)

Fakt

Gramatyka

$$S \to aAc,\ A \to bAb,\ A \to b$$

nie jest LR(k) dla żadnego k

Własności gramatyk LR(k)

Fakt

Gramatyka

$$S \to aAc,\ A \to bAb,\ A \to b$$

nie jest LR(k) dla żadnego k

Ćwiczenie: znaleźć równoważną gramatykę LR(0)

Własności gramatyk LR(k)

Fakt

Gramatyka

$$S \to aAc,\ A \to bAb,\ A \to b$$

nie jest LR(k) dla żadnego k

Ćwiczenie: znaleźć równoważną gramatykę LR(0)

Twierdzenie

Problem czy dla danej gramatyki G istnieje k takie, że $G \in LR(k)$ jest nierozstrzygalny.

Kiedy redukować?

Różne metody:

LR(0) — redukujemy kiedy się tylko da
w praktyce za słaba

LR(1) — precyzyjnie wyliczamy dla jakich terminali na wejściu redukować
bardzo silna metoda, ale koszt generowania rzędu 2^{n^2}.

SLR(1) — Simple LR(1): LR(0) + prosty pomysł: redukujemy $A \to \alpha$ jeśli terminal z wejścia należy do FOLLOW(A).

LALR(1) — Look Ahead LR(1): zgrubnie wyliczamy (budujemy automat LR(1) i sklejamy podobne stany).
w praktyce dostatecznie silna metoda,
tyle samo stanów co w automacie LR(0)

Jak rozpoznać uchwyt?

Zbudujemy automat skończony rozpoznający wiele wzorców (możliwe prawe strony produkcji)

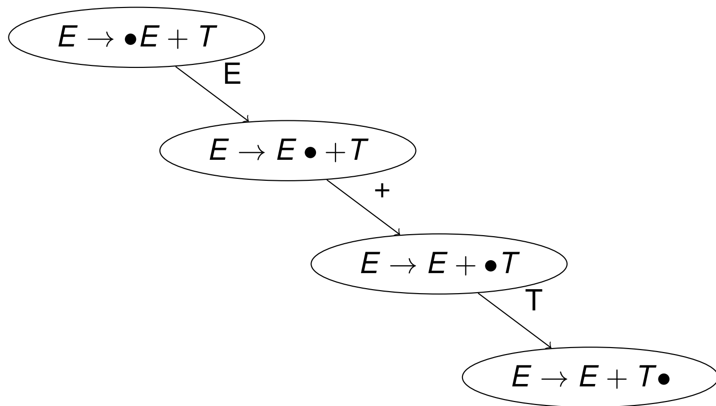
Sytuacja LR(0)

$$A \to \alpha \bullet \beta$$

czyli produkcja z wyróżnionym miejscem.
Jesteśmy w trakcie rozpoznawania $A \to \alpha\beta$,
na stosie jest już α, trzeba jeszcze rozpoznać β

Przykład (fragment automatu)

Jeśli mamy sytuację $A \to \alpha \bullet X\beta$, to po rozpoznaniu X mamy sytuację $A \to \alpha X \bullet \beta$



Stany i przejścia automatu LR

- Ponieważ trzeba równocześnie rozpoznawać wiele uchwytów, to stanami automatu będą **zbiory sytuacji.**
- Jeśli jesteśmy w sytuacji $B \to \alpha \bullet A\beta$, to jesteśmy też w sytuacji $A \to \bullet\gamma$ dla każdego $A \to \gamma \in P$
- Zbiór sytuacji musi być domknięty zwn tę implikację:
- $Closure(Q)$ – najmniejszy zbiór zawierający Q oraz taki, że jeśli $B \to \alpha \bullet A\beta \in Closure(Q)$,to

$$\forall A \to \gamma \in P \quad A \to \ \bullet\gamma \in Closure(Q)$$

- Jeśli $A \to \alpha \bullet X\gamma \in Q$ dla pewnego $X \in N \cup T$, to ze stanu Q jest przejście (po X) do stanu $Closure(A \to \alpha X \bullet \gamma)$

Przykład

$S \to E,\ E \to E + T \mid E - T \mid T,\ T \to T * F \mid F,\ F \to n$

Domknięciem stanu $S \to \bullet E$ jest stan zawierający również

Przykład

$S \to E,\ E \to E + T \mid E - T \mid T,\ T \to T * F \mid F,\ F \to n$

Domknięciem stanu $S \to \bullet E$ jest stan zawierający również

$$E \to \bullet E + T,\ E \to \bullet E - T,\ E \to \bullet T$$

Przykład

$$S \to E,\ E \to E + T \mid E - T \mid T,\ T \to T * F \mid F,\ F \to n$$

Domknięciem stanu $S \to \bullet E$ jest stan zawierający również

$$E \to \bullet E + T,\ E \to \bullet E - T,\ E \to \bullet T$$

$$T \to \bullet T * F,\ T \to \bullet F$$

Przykład

$$S \to E,\ E \to E + T \mid E - T \mid T,\ T \to T * F \mid F,\ F \to n$$

Domknięciem stanu $S \to \bullet E$ jest stan zawierający również

$$E \to \bullet E + T,\ E \to \bullet E - T,\ E \to \bullet T$$

$$T \to \bullet T * F,\ T \to \bullet F$$

$$F \to \bullet n$$

Przykład

$$S \to E,\ E \to E + T \mid E - T \mid T,\ T \to T * F \mid F,\ F \to n$$

Domknięciem stanu $S \to \bullet E$ jest stan zawierający również

$$E \to \bullet E + T,\ E \to \bullet E - T,\ E \to \bullet T$$

$$T \to \bullet T * F,\ T \to \bullet F$$

$$F \to \bullet n$$

Z tego stanu po rozpoznaniu E przejdziemy do stanu zawierającego domknięcie zbioru

$$S \to E \bullet,\ E \to E \bullet + T,\ E \to E \bullet - T$$

(ale już bez $E \to \bullet T$).

Które produkcje uwzględniać?

- Zaczynamy od jednej, nowej produkcji z nowym nieterminalem początkowym: $S' \to \bullet S\#$
- Dojście do sytuacji $S' \to S \bullet \#$ oznaczałoby szczęśliwy koniec pracy dla $\#$ na wejściu.
- Jeśli jesteśmy w sytuacji $B \to \alpha \bullet A\beta$, to trzeba zacząć rozpoznawać coś, co da się zredukować do A, czyli uwzględnić sytuacje postaci $A \to \bullet\gamma$.

Działanie automatu LR

- Dwie tablice indeksowane stanami i symbolami: ACTION (dla terminali) i GOTO (dla nieterminali)
- Stos zawiera stany przetykane symbolami gramatyki
- Automat startuje ze stosem zawierającym symbol początkowy
- Niech na stosie stan s, na wejściu terminal a:
  - ACTION[s, a] = **shift** p
    przenosi a z wejścia na stos i nakrywa stanem p
  - ACTION[s, a] = **reduce**($A \to \alpha$)
    zdejmuje $|\alpha|$ par ze stosu
    odsłoni się stan q (zawierał sytuację $\ldots \bullet A \ldots$)
    wkłada na stos A, GOTO[q, A].
  - Specjalne akcje: **error**, **accept**

Konstrukcja automatu LR

1. Rozszerzamy gramatykę o produkcję $S' \to S\#$ (nowy symbol początkowy)
2. Budujemy automat skończony:
   - stanami są zbiory sytuacji
   - stan początkowy: $Closure(\{S' \to \bullet S\#\})$
   - dla stanu p przejście po symbolu X do stanu

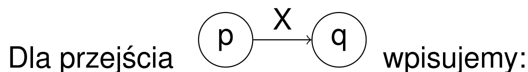
   $$\delta(p, X) = Closure(\{A \to \alpha X \bullet \gamma : A \to \alpha \bullet X\gamma \in p\})$$

   - stanem akceptującym jest $\{S' \to S\# \bullet\}$
3. Wypełniamy tablicę sterującą automatu ze stosem.

Przykład na tablicy

Wypełnianie tablic sterujących

Numerujemy stany, numerujemy produkcje.
Jednolicie dla wszystkich klas automatów wpisujemy akcje **shift** (przepisujemy przejścia automatu skończonego) i **accept**:

Dla przejścia  wpisujemy:

- jeśli X jest *terminalem* to

$$\mathrm{ACTION}[p, x] = \textbf{shift q}$$

- jeśli X jest *nieterminalem* to

$$\mathrm{GOTO}[p, x] = \textbf{q}$$

- Jeśli stan p zawiera $S' \to S \bullet \#$, to $\mathrm{ACTION}[p, \#] = \textbf{accept}$

Redukcje

Tu postępujemy różnie dla różnych klas automatów.

Jeśli stan p zawiera $A \to \alpha \bullet$, to:

- LR(0) wpisujemy **reduce**$(A \to \alpha)$ do ACTION$[p, a]$ dla wszystkich a
- SLR(1) wpisujemy **reduce**$(A \to \alpha)$ do ACTION$[p, a]$ dla $a \in \textsc{Follow}(A)$

Miejsca nie wypełnione oznaczają **error**
Jeśli gdzieś zostanie wpisana więcej niż jedna akcja, to źle: gramatyka nie jest odpowiedniej klasy (konflikt shift-reduce lub reduce-reduce).

Przykład na tablicy

LR(1), LALR(1) – na następnym wykładzie

Generatory parserów

- Ręczne tworzenie automatu LR jest w praktyce zbyt żmudne
- Może to z powodzeniem wykonać specjalny program
- Wejście: gramatyka translacyjna (gramatyke + akcje)
- Wyjście: analizator składniowy LR w języku docelowym
- Istnieją dla wielu języków: C,C++ (Yacc,Bison), Java (Cup), Haskell (Happy), Ocaml (Ocamlyacc), ...

Bison

Generatory parserów dla C/C++
Format wejścia:

```
deklaracje
%%
gramatyka atrybutywna: produkcje z akcjami postaci
a: b c { $$.x = f($1.y, $2.z);}
 | d { $$.x = $1.x; }
 ;
%%
dodatkowy kod (w C/C++)
```

Jeden atrybut, ale dowolnego typu (może być strukturą lub wskaźnikiem).
Wyjście: funkcja `int yyparse()` — w wyniku 0 jeśli sukces.
Wywołuje `yylex()` dla pobierania kolejnych leksemów.

```
%{
#define YYSTYPE double
%}
%token NUM
%%
input:     /* empty */
         | input line
         ;
line:      '\n'
         | exp '\n' { printf("\t%.10g\n", $1);}
         ;
exp:     NUM           { $$ = $1;       }
         | exp exp '+' { $$ = $1 + $2; }
         | exp exp '-' { $$ = $1 - $2; }
         | exp exp '*' { $$ = $1 * $2; }
         | exp exp '/' { $$ = $1 / $2; }
%%
```

Przykład deklaracji

```
%{
#include "parstree.h"
%}
%union {
tree_node_ptr node;       /* For parse tree nodes */
long           id ;          /* For identifiers */
long           num_lit ;     /* For number VALUES */
long           char_lit ;    /* For char literals */
long           string_lit ; /* For string literals *
}
%token <id>         TIDENTIFIER
%token <id>         TTYPEIDENTIFIER
%token <num_lit>    TINT_LITERAL
%token <char_lit>   TCHAR_LITERAL
%token <string_lit> TSTRING_LITERAL
%token TIF
...
```

Przykład deklaracji

```
...
%type <id>     id_or_empty
%type <node>   type_or_function_declarator
%type <node>   type_params
%type <node>   type_body
...
type_definition_clause :
  TTYPE TIDENTIFIER type_params '=' type_body ';'
  { $$ = mkptnode(TYPEDEF_CLAUSE, $3, $5, 0, $2);}
```

Java Cup

Podobnie jak bison, ale trochę inna składnia.
Cztery sekcje, nie rozdzielone niczym, ale muszą być w kolejności:

- `package`, `import...`
- Składniki kodu użytkownika
- Listy symboli (terminali i nieterminali)
- Gramatyka

Składniki kodu użytkownika

`parser code{: ... :}` — kod dołączany do klasy parsera
`init with{: ... :}` — kod który będzie wykonany zanim parser pobierze pierwszy leksem
`scan with{: ... :}` — kod do pobierania leksemów, wynik powinien być typu `java_cup.runtime.token`

Przykład

```
import java_cup.runtime.*;

init with {: scanner.init();                :};
scan with {: return scanner.next_token();  :};
```

Lista symboli

Należy zadeklarować wszystkie terminale i nieterminale:

```
terminal klasa nazwa1, nazwa2,...
non terminal klasa nazwa1, nazwa2,...
```

Jeżeli leksem nie ma wartości semantycznej, klasę opuszczamy.

Przykład:

```
terminal                SEMI, PLUS, MINUS;
terminal                TIMES, DIVIDE, MOD;
terminal                UMINUS, LPAREN, RPAREN;
terminal Integer        NUMBER;

non terminal                expr_list, expr_part;
non terminal Integer        expr, term, factor;
```

Gramatyka

Podobnie jak bison, ale trochę inna składnia:

```
expr_list ::= expr_list expr_part
      |
              expr_part;

expr_part ::= expr:e
      {: System.out.println("= " + e); :}
              SEMI
      ;
expr      ::= expr:e1 PLUS expr:e2
    {: RESULT = new Integer(e1.intValue()
                              + e2.intValue()); :}
        |
        expr:e1 MINUS expr:e2     ...
        | ...
```