Kompilátory

Cvičenie 4: Syntaktická analýza v ANTLR4 (1. časť)

Peter Kostolányi

7. novembra 2017

ANTLR: základné princípy (opakovanie)

ANTLR: základné princípy (opakovanie)

- Vstup: „gramatika“ v metajazyku ANTLR4

ANTLR: základné princípy (opakovanie)

- Vstup: „gramatika“ v metajazyku ANTLR4
- Typicky ide o súbor s príponou `.g4` alebo `.g`

ANTLR: základné princípy (opakovanie)

- Vstup: „gramatika“ v metajazyku ANTLR4
- Typicky ide o súbor s príponou `.g4` alebo `.g`

```
1  lexer grammar Vyrazy;

3  LEFT
       :     '('
5      ;

7  RIGHT
       :     ')'
9      ;

11 MUL
       :     '*'
13     ;

15 DIV
       :     '/'
17     ;

   ADD
20     :     '+'
       ;
22
   SUB
24     :     '-'
       ;
26
   INT
28     :     DIGIT+
       ;
30
   WS
32     :     [ \t\n\r]+   -> skip
       ;
34
   fragment DIGIT
36     :     [0-9]
       ;
```

ANTLR: základné princípy (opakovanie)

- Vstup: „gramatika“ v metajazyku ANTLR4
- Typicky ide o súbor s príponou `.g4` alebo `.g`

```
grammar Vyrazy;

expr
    :   expr op=(MUL|DIV) expr
    |   expr op=(ADD|SUB) expr
    |   '(' expr ')'
    |   INT
    ;

MUL
    :   '*'
    ;

DIV
    :   '/'
    ;

ADD
    :   '+'
    ;

SUB
    :   '-'
    ;

INT
    :   DIGIT+
    ;

WS
    :   [ \t\n\r]+   -> skip
    ;

fragment DIGIT
    :   [0-9]
    ;
```

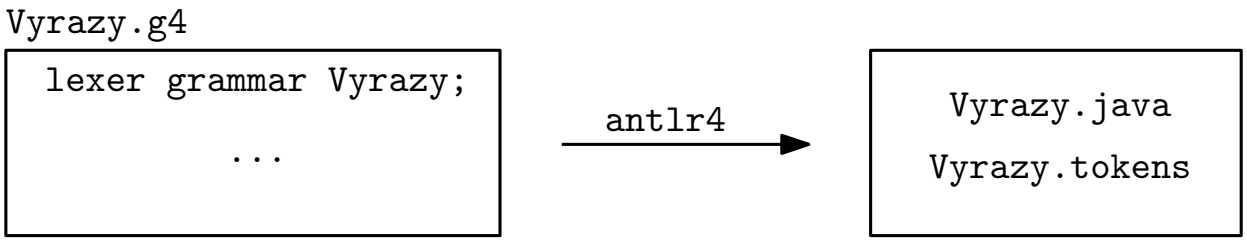
ANTLR: základné princípy (opakovanie)

Vyrazy.g4

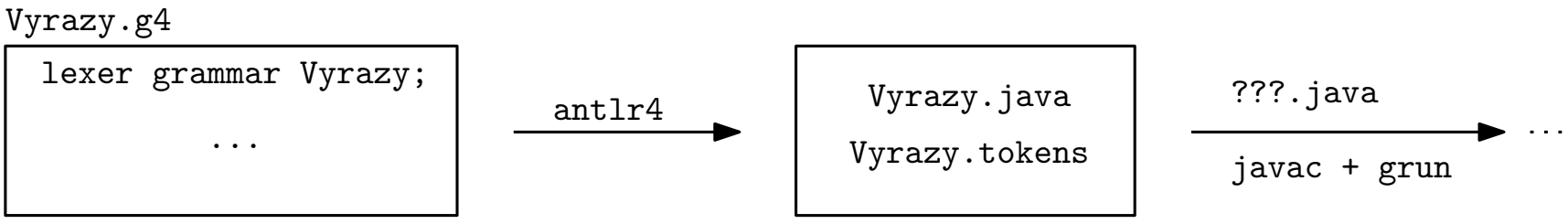
```
lexer grammar Vyrazy;

...
```

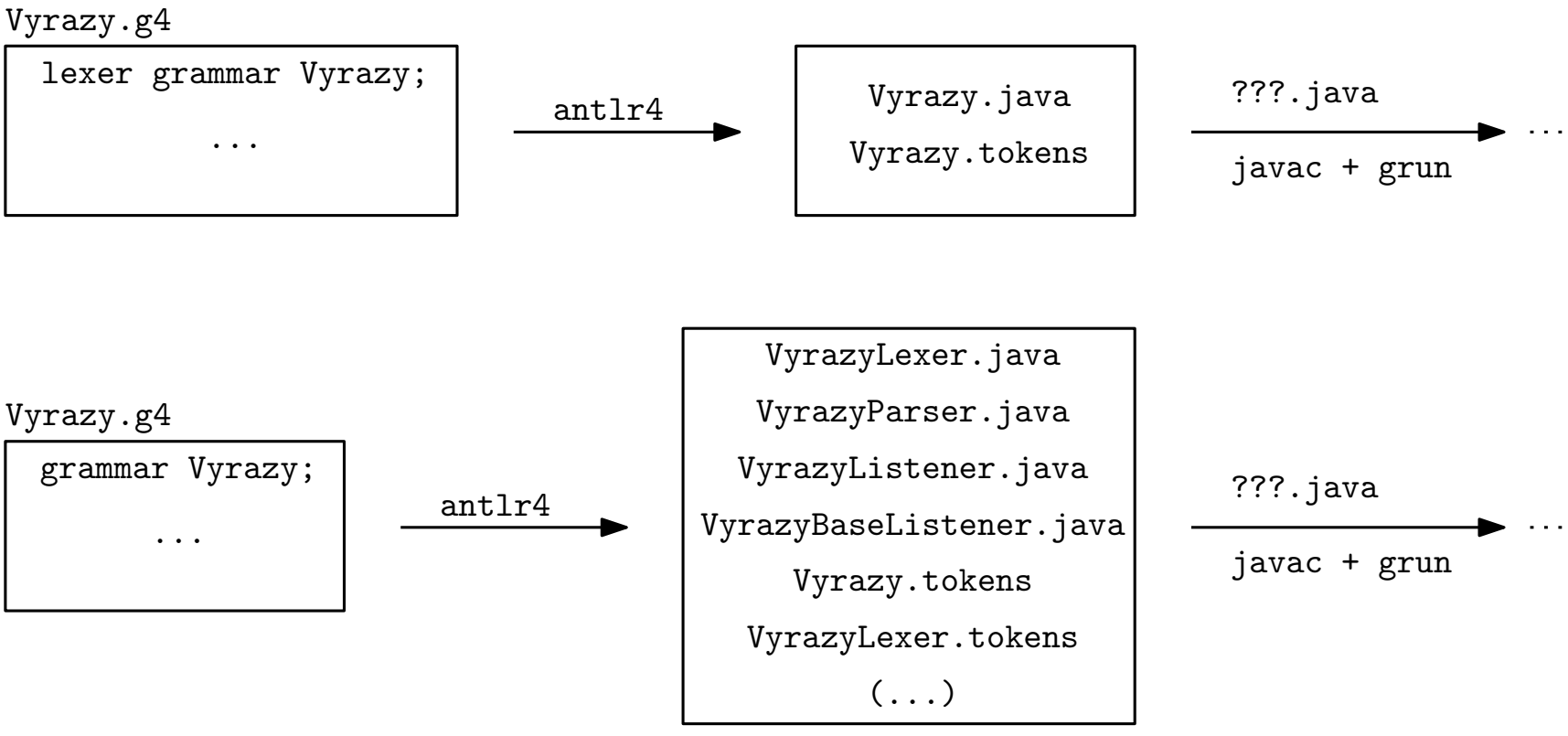
ANTLR: základné princípy (opakovanie)



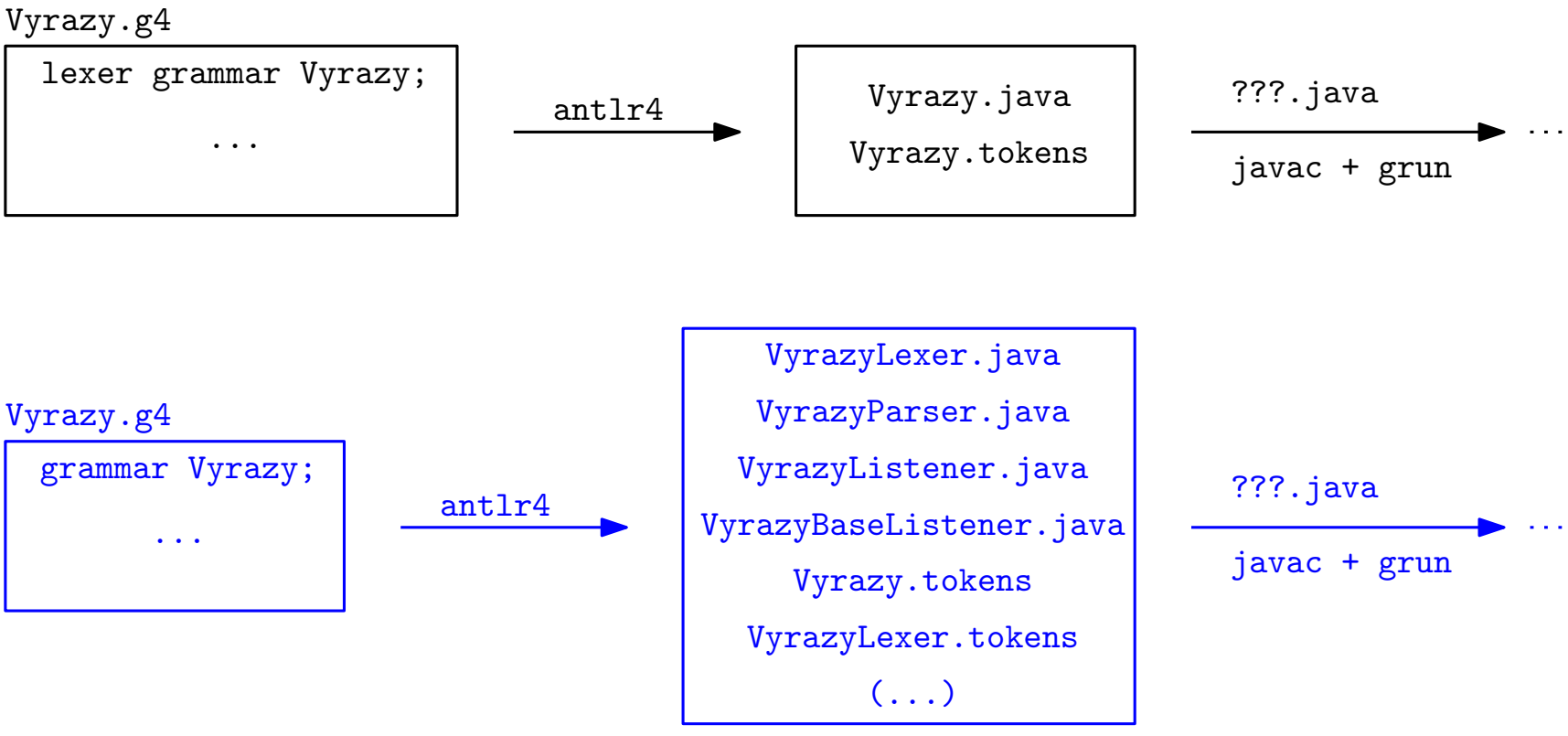
ANTLR: základné princípy (opakovanie)



ANTLR: základné princípy (opakovanie)



ANTLR: základné princípy (opakovanie)



Viacznačnosť a ľavá rekurzia

Uvažujme napríklad nasledujúcu časť gramatiky pre aritmetické výrazy:

Viacznačnosť a ľavá rekurzia

Uvažujme napríklad nasledujúcu časť gramatiky pre aritmetické výrazy:

```
1 expr
        :     expr (MUL|DIV) expr
3       |     expr (ADD|SUB) expr
        |     LEFT expr RIGHT
5       |     SUB LEFT expr RIGHT
        |     NUM
7       |     SUB NUM
        ;
```

Viacznačnosť a ľavá rekurzia

Uvažujme napríklad nasledujúcu časť gramatiky pre aritmetické výrazy:

```
1 expr
        :     expr (MUL|DIV) expr
3       |     expr (ADD|SUB) expr
        |     LEFT expr RIGHT
5       |     SUB LEFT expr RIGHT
        |     NUM
7       |     SUB NUM
        ;
```

- Takáto definícia je viacznačná

Viacznačnosť a ľavá rekurzia

Uvažujme napríklad nasledujúcu časť gramatiky pre aritmetické výrazy:

```
1 expr
        :     expr (MUL|DIV) expr
3       |     expr (ADD|SUB) expr
        |     LEFT expr RIGHT
5       |     SUB LEFT expr RIGHT
        |     NUM
7       |     SUB NUM
        ;
```

- Takáto definícia je viacznačná
- Existujú napríklad dve odvodenia pre $2 + 2 * 2$

Viacznačnosť a ľavá rekurzia

Uvažujme napríklad nasledujúcu časť gramatiky pre aritmetické výrazy:

```
1 expr
        :     expr (MUL|DIV) expr
3       |     expr (ADD|SUB) expr
        |     LEFT expr RIGHT
5       |     SUB LEFT expr RIGHT
        |     NUM
7       |     SUB NUM
        ;
```

- Takáto definícia je viacznačná
- Existujú napríklad dve odvodenia pre $2 + 2 * 2$
- Definícia je tiež ľavo rekurzívna, čo pri parsovaní zhora nadol predstavuje problém

Viacznačnosť a ľavá rekurzia

ANTLR4 vie automaticky odstrániť bezprostrednú ľavú rekurziu:

Viacznačnosť a ľavá rekurzia

ANTLR4 vie automaticky odstrániť bezprostrednú ľavú rekurziu:

- Tá postačuje na popis väčšiny v praxi sa vyskytujúcich konštrukcií

Viacznačnosť a ľavá rekurzia

ANTLR4 vie automaticky odstrániť bezprostrednú ľavú rekurziu:

- Tá postačuje na popis väčšiny v praxi sa vyskytujúcich konštrukcií
- Viac o tejto transformácii na budúcom cvičení

Viacznačnosť a ľavá rekurzia

ANTLR4 vie automaticky odstrániť bezprostrednú ľavú rekurziu:

- Tá postačuje na popis väčšiny v praxi sa vyskytujúcich konštrukcií
- Viac o tejto transformácii na budúcom cvičení

Automatické „zjednoznačnenie“ viacznačnej gramatiky:

Viacznačnosť a ľavá rekurzia

ANTLR4 vie automaticky odstrániť bezprostrednú ľavú rekurziu:

- Tá postačuje na popis väčšiny v praxi sa vyskytujúcich konštrukcií
- Viac o tejto transformácii na budúcom cvičení

Automatické „zjednoznačnenie“ viacznačnej gramatiky:

- Vhodné pre gramatiky popisujúce nejaký druh výrazov

Viacznačnosť a ľavá rekurzia

ANTLR4 vie automaticky odstrániť bezprostrednú ľavú rekurziu:

- Tá postačuje na popis väčšiny v praxi sa vyskytujúcich konštrukcií
- Viac o tejto transformácii na budúcom cvičení

Automatické „zjednoznačnenie“ viacznačnej gramatiky:

- Vhodné pre gramatiky popisujúce nejaký druh výrazov
- Používa metódu známu ako „precedence climbing“

Viacznačnosť a ľavá rekurzia

ANTLR4 vie automaticky odstrániť bezprostrednú ľavú rekurziu:

- Tá postačuje na popis väčšiny v praxi sa vyskytujúcich konštrukcií
- Viac o tejto transformácii na budúcom cvičení

Automatické „zjednoznačnenie“ viacznačnej gramatiky:

- Vhodné pre gramatiky popisujúce nejaký druh výrazov
- Používa metódu známu ako „precedence climbing“
- Pre konštrukcie iné ako výrazy je odporúčané riešiť viacznačnosť manuálne (zmenou gramatiky alebo pridaním predikátov)

Viacznačnosť a ľavá rekurzia

ANTLR4 vie automaticky odstrániť bezprostrednú ľavú rekurziu:

- Tá postačuje na popis väčšiny v praxi sa vyskytujúcich konštrukcií
- Viac o tejto transformácii na budúcom cvičení

Automatické „zjednoznačnenie“ viacznačnej gramatiky:

- Vhodné pre gramatiky popisujúce nejaký druh výrazov
- Používa metódu známu ako „precedence climbing“
- Pre konštrukcie iné ako výrazy je odporúčané riešiť viacznačnosť manuálne (zmenou gramatiky alebo pridaním predikátov)
- ANTLR4 identifikuje štyri typy „rekurzívnych vzorov“:

Viacznačnosť a ľavá rekurzia

ANTLR4 vie automaticky odstrániť bezprostrednú ľavú rekurziu:

- Tá postačuje na popis väčšiny v praxi sa vyskytujúcich konštrukcií
- Viac o tejto transformácii na budúcom cvičení

Automatické „zjednoznačnenie“ viacznačnej gramatiky:

- Vhodné pre gramatiky popisujúce nejaký druh výrazov
- Používa metódu známu ako „precedence climbing“
- Pre konštrukcie iné ako výrazy je odporúčané riešiť viacznačnosť manuálne (zmenou gramatiky alebo pridaním predikátov)
- ANTLR4 identifikuje štyri typy „rekurzívnych vzorov“:

  Binárny: `expr op expr`

Viacznačnosť a ľavá rekurzia

ANTLR4 vie automaticky odstrániť bezprostrednú ľavú rekurziu:

- Tá postačuje na popis väčšiny v praxi sa vyskytujúcich konštrukcií
- Viac o tejto transformácii na budúcom cvičení

Automatické „zjednoznačnenie“ viacznačnej gramatiky:

- Vhodné pre gramatiky popisujúce nejaký druh výrazov
- Používa metódu známu ako „precedence climbing“
- Pre konštrukcie iné ako výrazy je odporúčané riešiť viacznačnosť manuálne (zmenou gramatiky alebo pridaním predikátov)
- ANTLR4 identifikuje štyri typy „rekurzívnych vzorov“:

  Binárny: `expr op expr`

  Ternárny: `expr op1 expr op2 expr`

Viacznačnosť a ľavá rekurzia

ANTLR4 vie automaticky odstrániť bezprostrednú ľavú rekurziu:

- Tá postačuje na popis väčšiny v praxi sa vyskytujúcich konštrukcií
- Viac o tejto transformácii na budúcom cvičení

Automatické „zjednoznačnenie“ viacznačnej gramatiky:

- Vhodné pre gramatiky popisujúce nejaký druh výrazov
- Používa metódu známu ako „precedence climbing“
- Pre konštrukcie iné ako výrazy je odporúčané riešiť viacznačnosť manuálne (zmenou gramatiky alebo pridaním predikátov)
- ANTLR4 identifikuje štyri typy „rekurzívnych vzorov“:

  Binárny: `expr op expr`
  Ternárny: `expr op1 expr op2 expr`
  Prefixový: `niečo expr`

Viacznačnosť a ľavá rekurzia

ANTLR4 vie automaticky odstrániť bezprostrednú ľavú rekurziu:

- Tá postačuje na popis väčšiny v praxi sa vyskytujúcich konštrukcií
- Viac o tejto transformácii na budúcom cvičení

Automatické „zjednoznačnenie“ viacznačnej gramatiky:

- Vhodné pre gramatiky popisujúce nejaký druh výrazov
- Používa metódu známu ako „precedence climbing“
- Pre konštrukcie iné ako výrazy je odporúčané riešiť viacznačnosť manuálne (zmenou gramatiky alebo pridaním predikátov)
- ANTLR4 identifikuje štyri typy „rekurzívnych vzorov“:

  Binárny: `expr op expr`
  Ternárny: `expr op1 expr op2 expr`
  Prefixový: `niečo expr`
  Sufixový: `expr niečo`

Viacznačnosť a ľavá rekurzia

ANTLR4 vie automaticky odstrániť bezprostrednú ľavú rekurziu:

- Tá postačuje na popis väčšiny v praxi sa vyskytujúcich konštrukcií
- Viac o tejto transformácii na budúcom cvičení

Automatické „zjednoznačnenie“ viacznačnej gramatiky:

- Vhodné pre gramatiky popisujúce nejaký druh výrazov
- Používa metódu známu ako „precedence climbing“
- Pre konštrukcie iné ako výrazy je odporúčané riešiť viacznačnosť manuálne (zmenou gramatiky alebo pridaním predikátov)
- ANTLR4 identifikuje štyri typy „rekurzívnych vzorov“:

  Binárny: `expr op expr`
  Ternárny: `expr op1 expr op2 expr`
  Prefixový: `niečo expr`
  Sufixový: `expr niečo`

- Skôr uvedená alternatíva $\rightsquigarrow$ vyššia precedencia operátorov

Viacznačnosť a ľavá rekurzia

ANTLR4 vie automaticky odstrániť bezprostrednú ľavú rekurziu:

- Tá postačuje na popis väčšiny v praxi sa vyskytujúcich konštrukcií
- Viac o tejto transformácii na budúcom cvičení

Automatické „zjednoznačnenie“ viacznačnej gramatiky:

- Vhodné pre gramatiky popisujúce nejaký druh výrazov
- Používa metódu známu ako „precedence climbing“
- Pre konštrukcie iné ako výrazy je odporúčané riešiť viacznačnosť manuálne (zmenou gramatiky alebo pridaním predikátov)
- ANTLR4 identifikuje štyri typy „rekurzívnych vzorov“:

  Binárny: `expr op expr`
  Ternárny: `expr op1 expr op2 expr`
  Prefixový: `niečo expr`
  Sufixový: `expr niečo`

- Skôr uvedená alternatíva $\rightsquigarrow$ vyššia precedencia operátorov
- Alternatívy nezodpovedajúce žiadnemu vzoru sa považujú za „primárne výrazy“:

Viacznačnosť a ľavá rekurzia

ANTLR4 vie automaticky odstrániť bezprostrednú ľavú rekurziu:

- Tá postačuje na popis väčšiny v praxi sa vyskytujúcich konštrukcií
- Viac o tejto transformácii na budúcom cvičení

Automatické „zjednoznačnenie“ viacznačnej gramatiky:

- Vhodné pre gramatiky popisujúce nejaký druh výrazov
- Používa metódu známu ako „precedence climbing“
- Pre konštrukcie iné ako výrazy je odporúčané riešiť viacznačnosť manuálne (zmenou gramatiky alebo pridaním predikátov)
- ANTLR4 identifikuje štyri typy „rekurzívnych vzorov“:

  Binárny: `expr op expr`
  Ternárny: `expr op1 expr op2 expr`
  Prefixový: `niečo expr`
  Sufixový: `expr niečo`

- Skôr uvedená alternatíva $\rightsquigarrow$ vyššia precedencia operátorov
- Alternatívy nezodpovedajúce žiadnemu vzoru sa považujú za „primárne výrazy“:
  - `NUM`

Viacznačnosť a ľavá rekurzia

ANTLR4 vie automaticky odstrániť bezprostrednú ľavú rekurziu:

- Tá postačuje na popis väčšiny v praxi sa vyskytujúcich konštrukcií
- Viac o tejto transformácii na budúcom cvičení

Automatické „zjednoznačnenie“ viacznačnej gramatiky:

- Vhodné pre gramatiky popisujúce nejaký druh výrazov
- Používa metódu známu ako „precedence climbing“
- Pre konštrukcie iné ako výrazy je odporúčané riešiť viacznačnosť manuálne (zmenou gramatiky alebo pridaním predikátov)
- ANTLR4 identifikuje štyri typy „rekurzívnych vzorov“:

  Binárny: `expr op expr`
  Ternárny: `expr op1 expr op2 expr`
  Prefixový: `niečo expr`
  Sufixový: `expr niečo`

- Skôr uvedená alternatíva $\rightsquigarrow$ vyššia precedencia operátorov
- Alternatívy nezodpovedajúce žiadnemu vzoru sa považujú za „primárne výrazy“:
  - `NUM`
  - `LEFT expr RIGHT`

Viacznačnosť a ľavá rekurzia

ANTLR4 vie automaticky odstrániť bezprostrednú ľavú rekurziu:

- Tá postačuje na popis väčšiny v praxi sa vyskytujúcich konštrukcií
- Viac o tejto transformácii na budúcom cvičení

Automatické „zjednoznačnenie“ viacznačnej gramatiky:

- Vhodné pre gramatiky popisujúce nejaký druh výrazov
- Používa metódu známu ako „precedence climbing“
- Pre konštrukcie iné ako výrazy je odporúčané riešiť viacznačnosť manuálne (zmenou gramatiky alebo pridaním predikátov)
- ANTLR4 identifikuje štyri typy „rekurzívnych vzorov“:

  Binárny: `expr op expr`
  Ternárny: `expr op1 expr op2 expr`
  Prefixový: `niečo expr`
  Sufixový: `expr niečo`

- Skôr uvedená alternatíva $\rightsquigarrow$ vyššia precedencia operátorov
- Alternatívy nezodpovedajúce žiadnemu vzoru sa považujú za „primárne výrazy“:
  - `NUM`
  - `LEFT expr RIGHT`
  - `...`

Použitie ANTLR4 a nástroja TestRig

Gramatiku možno otestovať volaním `grun` s vhodnými parametrami:

Použitie ANTLR4 a nástroja TestRig

Gramatiku možno otestovať volaním grun s vhodnými parametrami:

```
1 antlr4 NazovGramatiky.g4
2 javac *.java
3 grun NazovGramatiky pociatocnyNeterminal -tokens
4 grun NazovGramatiky pociatocnyNeterminal -tree
5 grun NazovGramatiky pociatocnyNeterminal -ps strom.ps
6 grun NazovGramatiky pociatocnyNeterminal -gui
7 grun NazovGramatiky pociatocnyNeterminal -gui vstup.in
```

Importovanie gramatík

- Veľké gramatiky je často vhodné rozdeliť na viac častí

Importovanie gramatík

- Veľké gramatiky je často vhodné rozdeliť na viac častí
- Príkaz na importovanie pravidiel z inej gramatiky:
  `import Gramatika;`

Importovanie gramatík

- Veľké gramatiky je často vhodné rozdeliť na viac častí
- Príkaz na importovanie pravidiel z inej gramatiky:
  `import Gramatika;`
- Vhodné napríklad na oddelenie syntaktických a lexikálnych pravidiel

Importovanie gramatík

- Veľké gramatiky je často vhodné rozdeliť na viac častí
- Príkaz na importovanie pravidiel z inej gramatiky:
  `import Gramatika;`
- Vhodné napríklad na oddelenie syntaktických a lexikálnych pravidiel
- V prípade konfliktu má definícia v importujúcej gramatike prednosť pred definíciou v importovanej gramatike

Integrácia vygenerovaného parsera do vlastného programu

Základ programu využívajúceho vygenerovaný parser:

Integrácia vygenerovaného parsera do vlastného programu

Základ programu využívajúceho vygenerovaný parser:

```
1  import org.antlr.v4.runtime.*;
   import org.antlr.v4.runtime.tree.*;
3  import java.io.*;
   import java.util.*;
5
   public class Hlavna {
7
       public static void main(String[] args) throws Exception {
9          InputStream is = System.in;
           if (args.length > 0) {
11             is = new FileInputStream(args[0]);
           }
13         CharStream input = CharStreams.fromStream(is);
           GramatikaLexer lexer = new GramatikaLexer(input);
15         CommonTokenStream tokens = new CommonTokenStream(lexer);
           GramatikaParser parser = new GramatikaParser(tokens);
17
           ParseTree tree = parser.pociatocnyNeterminal();
19     }

21 }
```

„Kontexty“ neterminálov

Pre každý neterminál `neterminal` obsahuje trieda `GramatikaParser` statickú podtriedu `NeterminalContext`:

„Kontexty“ neterminálov

Pre každý neterminál `neterminal` obsahuje trieda `GramatikaParser` statickú podtriedu `NeterminalContext`:

- Inštancia takejto triedy reprezentuje „výskyt“ daného neterminálu v syntaktickom strome

„Kontexty“ neterminálov

Pre každý neterminál `neterminal` obsahuje trieda `GramatikaParser` statickú podtriedu `NeterminalContext`:

- Inštancia takejto triedy reprezentuje „výskyt“ daného neterminálu v syntaktickom strome
- Každá z tried `NeterminalContext` dedí od `ParserRuleContext`

„Kontexty“ neterminálov

Pre každý neterminál `neterminal` obsahuje trieda `GramatikaParser` statickú podtriedu `NeterminalContext`:

- Inštancia takejto triedy reprezentuje „výskyt“ daného neterminálu v syntaktickom strome
- Každá z tried `NeterminalContext` dedí od `ParserRuleContext`
- `ParserRuleContext` okrem iného implementuje `ParseTree`

„Kontexty“ neterminálov

Pre každý neterminál `neterminal` obsahuje trieda `GramatikaParser` statickú podtriedu `NeterminalContext`:

- Inštancia takejto triedy reprezentuje „výskyt“ daného neterminálu v syntaktickom strome
- Každá z tried `NeterminalContext` dedí od `ParserRuleContext`
- `ParserRuleContext` okrem iného implementuje `ParseTree`
- Na „kontext“ sa teda možno dívať aj ako na celý podstrom zakorenený v danom neterminál

„Kontexty“ neterminálov

Pre každý neterminál `neterminal` obsahuje trieda `GramatikaParser` statickú podtriedu `NeterminalContext`:

- Inštancia takejto triedy reprezentuje „výskyt“ daného neterminálu v syntaktickom strome
- Každá z tried `NeterminalContext` dedí od `ParserRuleContext`
- `ParserRuleContext` okrem iného implementuje `ParseTree`
- Na „kontext“ sa teda možno dívať aj ako na celý podstrom zakorenený v danom neterminál
- Metódy `neterminal()` definované v `GramatikaParser` vracajú inštancie triedy `NeterminalContext`

„Kontexty“ neterminálov

Pre každý neterminál `neterminal` obsahuje trieda `GramatikaParser` statickú podtriedu `NeterminalContext`:

- Inštancia takejto triedy reprezentuje „výskyt“ daného neterminálu v syntaktickom strome
- Každá z tried `NeterminalContext` dedí od `ParserRuleContext`
- `ParserRuleContext` okrem iného implementuje `ParseTree`
- Na „kontext“ sa teda možno dívať aj ako na celý podstrom zakorenený v danom neterminál
- Metódy `neterminal()` definované v `GramatikaParser` vracajú inštancie triedy `NeterminalContext`
- Triedy `NeterminalContext` majú definované metódy na prístup k ich deťom (ich definície možno nájsť priamo v súbore `GramatikaParser.java`)

Traverzovanie syntaktického stromu: „Listener“

- ANTLR4 Runtime API obsahuje triedu `ParseTreeWalker`

Traverzovanie syntaktického stromu: „Listener“

- ANTLR4 Runtime API obsahuje triedu `ParseTreeWalker`
- Tá obsahuje metódu, ktorá pretraverzuje strom prehľadávaním do hĺbky (vždy pre „najľavejšie“ dieťa):
  `void walk(ParseTreeListener listener, ParseTree t)`

Traverzovanie syntaktického stromu: „Listener“

- ANTLR4 Runtime API obsahuje triedu `ParseTreeWalker`
- Tá obsahuje metódu, ktorá pretraverzuje strom prehľadávaním do hĺbky (vždy pre „najľavejšie“ dieťa):
  `void walk(ParseTreeListener listener, ParseTree t)`

Volaním `antlr4 Gramatika.g4` sú okrem iného vytvorené:

Traverzovanie syntaktického stromu: „Listener“

- ANTLR4 Runtime API obsahuje triedu `ParseTreeWalker`
- Tá obsahuje metódu, ktorá pretraverzuje strom prehľadávaním do hĺbky (vždy pre „najľavejšie“ dieťa):
  `void walk(ParseTreeListener listener, ParseTree t)`

Volaním `antlr4 Gramatika.g4` sú okrem iného vytvorené:

- Rozhranie `GramatikaListener` dediace od `ParseTreeListener`

Traverzovanie syntaktického stromu: „Listener“

- ANTLR4 Runtime API obsahuje triedu `ParseTreeWalker`
- Tá obsahuje metódu, ktorá pretraverzuje strom prehľadávaním do hĺbky (vždy pre „najľavejšie“ dieťa):
  `void walk(ParseTreeListener listener, ParseTree t)`

Volaním `antlr4 Gramatika.g4` sú okrem iného vytvorené:

- Rozhranie `GramatikaListener` dediace od `ParseTreeListener`
- Trieda `GramatikaBaseListener` implementujúca `GramatikaListener`

Traverzovanie syntaktického stromu: „Listener“

- ANTLR4 Runtime API obsahuje triedu `ParseTreeWalker`
- Tá obsahuje metódu, ktorá pretraverzuje strom prehľadávaním do hĺbky (vždy pre „najľavejšie“ dieťa):
  `void walk(ParseTreeListener listener, ParseTree t)`

Volaním `antlr4 Gramatika.g4` sú okrem iného vytvorené:

- Rozhranie `GramatikaListener` dediace od `ParseTreeListener`
- Trieda `GramatikaBaseListener` implementujúca `GramatikaListener`
- `GramatikaBaseListener` obsahuje východzie (prázdne) implementácie metód reagujúcich na „objavenie“ a „opustenie“ vrchola stromu

Traverzovanie syntaktického stromu: „Listener“

- ANTLR4 Runtime API obsahuje triedu `ParseTreeWalker`
- Tá obsahuje metódu, ktorá pretraverzuje strom prehľadávaním do hĺbky (vždy pre „najľavejšie“ dieťa):
  `void walk(ParseTreeListener listener, ParseTree t)`

Volaním `antlr4 Gramatika.g4` sú okrem iného vytvorené:

- Rozhranie `GramatikaListener` dediace od `ParseTreeListener`
- Trieda `GramatikaBaseListener` implementujúca `GramatikaListener`
- `GramatikaBaseListener` obsahuje východzie (prázdne) implementácie metód reagujúcich na „objavenie“ a „opustenie“ vrchola stromu
- Od `GramatikaBaseListener` môže dediť trieda obsahujúca ozajstné implementácie týchto metód

Traverzovanie syntaktického stromu: „Listener“

- ANTLR4 Runtime API obsahuje triedu `ParseTreeWalker`
- Tá obsahuje metódu, ktorá pretraverzuje strom prehľadávaním do hĺbky (vždy pre „najľavejšie“ dieťa):
  `void walk(ParseTreeListener listener, ParseTree t)`

Volaním `antlr4 Gramatika.g4` sú okrem iného vytvorené:

- Rozhranie `GramatikaListener` dediace od `ParseTreeListener`
- Trieda `GramatikaBaseListener` implementujúca `GramatikaListener`
- `GramatikaBaseListener` obsahuje východzie (prázdne) implementácie metód reagujúcich na „objavenie“ a „opustenie“ vrchola stromu
- Od `GramatikaBaseListener` môže dediť trieda obsahujúca ozajstné implementácie týchto metód
- Vďaka rozdeleniu na rozhranie a „bazálnu“ triedu stačí implementovať metódy, kde treba niečo robiť

Traverzovanie syntaktického stromu: „Listener“

- ANTLR4 Runtime API obsahuje triedu `ParseTreeWalker`
- Tá obsahuje metódu, ktorá pretraverzuje strom prehľadávaním do hĺbky (vždy pre „najľavejšie“ dieťa):
  `void walk(ParseTreeListener listener, ParseTree t)`

Volaním `antlr4 Gramatika.g4` sú okrem iného vytvorené:

- Rozhranie `GramatikaListener` dediace od `ParseTreeListener`
- Trieda `GramatikaBaseListener` implementujúca `GramatikaListener`
- `GramatikaBaseListener` obsahuje východzie (prázdne) implementácie metód reagujúcich na „objavenie“ a „opustenie“ vrchola stromu
- Od `GramatikaBaseListener` môže dediť trieda obsahujúca ozajstné implementácie týchto metód
- Vďaka rozdeleniu na rozhranie a „bazálnu“ triedu stačí implementovať metódy, kde treba niečo robiť
- Pozor: triedu `GramatikaBaseListener` využívať len na dedenie („premaže“ sa pri každom volaní `antlr4`)

Označenie alternatív

Alternatívy pre každý neterminál možno „označiť“:

Označenie alternatív

Alternatívy pre každý neterminál možno „označiť“:

```
1 expr
      :     expr (MUL|DIV) expr     # BinOp
3     |     expr (ADD|SUB) expr     # BinOp
      |     LEFT expr RIGHT         # Paren
5     |     SUB LEFT expr RIGHT     # NegParen
      |     NUM                     # Number
7     |     SUB NUM                 # NegNumber
      ;
```

Označenie alternatív

Alternatívy pre každý neterminál možno „označiť“:

```
1 expr
      :     expr (MUL|DIV) expr     # BinOp
3     |     expr (ADD|SUB) expr     # BinOp
      |     LEFT expr RIGHT         # Paren
5     |     SUB LEFT expr RIGHT     # NegParen
      |     NUM                     # Number
7     |     SUB NUM                 # NegNumber
      ;
```

- „Kontexty“ sú vytvorené ku každej „značke“

Označenie alternatív

Alternatívy pre každý neterminál možno „označiť“:

```
1 expr
      :     expr (MUL|DIV) expr     # BinOp
3     |     expr (ADD|SUB) expr     # BinOp
      |     LEFT expr RIGHT         # Paren
5     |     SUB LEFT expr RIGHT     # NegParen
      |     NUM                     # Number
7     |     SUB NUM                 # NegNumber
      ;
```

- „Kontexty“ sú vytvorené ku každej „značke“
- „Listener“ potom tiež obsahuje metódy pre každý „kontext“

Označenie alternatív

Alternatívy pre každý neterminál možno „označiť“:

```
expr
    :   expr (MUL|DIV) expr     # BinOp
    |   expr (ADD|SUB) expr     # BinOp
    |   LEFT expr RIGHT         # Paren
    |   SUB LEFT expr RIGHT     # NegParen
    |   NUM                     # Number
    |   SUB NUM                 # NegNumber
    ;
```

- „Kontexty“ sú vytvorené ku každej „značke“
- „Listener“ potom tiež obsahuje metódy pre každý „kontext“
- Pre každý neterminál treba označiť buď všetky alternatívy, alebo žiadnu alternatívu

Pomenovanie prvkov pravidla

- Prvky pravidla možno pomenovať pre jednoduchšiu referenciu

Pomenovanie prvkov pravidla

- Prvky pravidla možno pomenovať pre jednoduchšiu referenciu
- Typicky ide o pomenovanie jedného výskytu neterminálu alebo alternatívy medzi tokenmi

Pomenovanie prvkov pravidla

- Prvky pravidla možno pomenovať pre jednoduchšiu referenciu
- Typicky ide o pomenovanie jedného výskytu neterminálu alebo alternatívy medzi tokenmi
- K danému pomenovaniu je následne vytvorený atribút zodpovedajúcej „kontextovej“ triedy (ide o inštanciu inej „kontextovej“ triedy alebo o inštanciu triedy `Token`)

Pomenovanie prvkov pravidla

- Prvky pravidla možno pomenovať pre jednoduchšiu referenciu
- Typicky ide o pomenovanie jedného výskytu neterminálu alebo alternatívy medzi tokenmi
- K danému pomenovaniu je následne vytvorený atribút zodpovedajúcej „kontextovej“ triedy (ide o inštanciu inej „kontextovej“ triedy alebo o inštanciu triedy `Token`)

```
1 expr
      :     expr op=(MUL|DIV) expr          # BinOp
3     |     expr op=(ADD|SUB) expr          # BinOp
      |     LEFT expr RIGHT                 # Paren
5     |     SUB LEFT expr RIGHT             # NegParen
      |     NUM                             # Number
7     |     SUB NUM                         # NegNumber
      ;
```

Traverzovanie syntaktického stromu: „Visitor“

- „Silnejšia“ metóda traverzovania stromov

Traverzovanie syntaktického stromu: „Visitor“

- „Silnejšia“ metóda traverzovania stromov
- Navštívené podstromy sú špecifikované programátorom (už teda nemusí ísť len o prehľadávanie do hĺbky)

Traverzovanie syntaktického stromu: „Visitor“

- „Silnejšia“ metóda traverzovania stromov
- Navštívené podstromy sú špecifikované programátorom (už teda nemusí ísť len o prehľadávanie do hĺbky)
- Metódy na návštevu podstromov vracajú na výstupe objekty nejakého typu

Traverzovanie syntaktického stromu: „Visitor“

- „Silnejšia“ metóda traverzovania stromov
- Navštívené podstromy sú špecifikované programátorom (už teda nemusí ísť len o prehľadávanie do hĺbky)
- Metódy na návštevu podstromov vracajú na výstupe objekty nejakého typu
- To umožňuje implementovať vyhodnocovanie atribútov a podobne

Traverzovanie syntaktického stromu: „Visitor“

- „Silnejšia“ metóda traverzovania stromov
- Navštívené podstromy sú špecifikované programátorom (už teda nemusí ísť len o prehľadávanie do hĺbky)
- Metódy na návštevu podstromov vracajú na výstupe objekty nejakého typu
- To umožňuje implementovať vyhodnocovanie atribútov a podobne

Na vytvorenie tried pre „Visitor“ namiesto tried pre „Listener“:

Traverzovanie syntaktického stromu: „Visitor“

- „Silnejšia“ metóda traverzovania stromov
- Navštívené podstromy sú špecifikované programátorom (už teda nemusí ísť len o prehľadávanie do hĺbky)
- Metódy na návštevu podstromov vracajú na výstupe objekty nejakého typu
- To umožňuje implementovať vyhodnocovanie atribútov a podobne

Na vytvorenie tried pre „Visitor“ namiesto tried pre „Listener“:

```
1 antlr4 -visitor -no-listener NazovGramatiky.g4
```

Traverzovanie syntaktického stromu: „Visitor“

Uvedeným volaním sú okrem iného vytvorené:

Traverzovanie syntaktického stromu: „Visitor“

Uvedeným volaním sú okrem iného vytvorené:

- Rozhranie `GramatikaVisitor<T>` dediace od `ParseTreeVisitor<T>`

Traverzovanie syntaktického stromu: „Visitor“

Uvedeným volaním sú okrem iného vytvorené:

- Rozhranie `GramatikaVisitor<T>` dediace od `ParseTreeVisitor<T>`
- Trieda `GramatikaBaseVisitor<T>` dediaca od `AbstractParseTreeVisitor<T>` a implementujúca `GramatikaVisitor<T>`

Traverzovanie syntaktického stromu: „Visitor“

Uvedeným volaním sú okrem iného vytvorené:

- Rozhranie `GramatikaVisitor<T>` dediace od `ParseTreeVisitor<T>`
- Trieda `GramatikaBaseVisitor<T>` dediaca od `AbstractParseTreeVisitor<T>` a implementujúca `GramatikaVisitor<T>`
- `GramatikaBaseVisitor<T>` obsahuje východzie implementácie metód na návštevu koreňa podstromu (navštív všetky deti)

Traverzovanie syntaktického stromu: „Visitor“

Uvedeným volaním sú okrem iného vytvorené:

- Rozhranie `GramatikaVisitor<T>` dediace od `ParseTreeVisitor<T>`
- Trieda `GramatikaBaseVisitor<T>` dediaca od `AbstractParseTreeVisitor<T>` a implementujúca `GramatikaVisitor<T>`
- `GramatikaBaseVisitor<T>` obsahuje východzie implementácie metód na návštevu koreňa podstromu (navštív všetky deti)
- Od `GramatikaBaseVisitor<T>` môže dediť trieda obsahujúca ozajstné implementácie týchto metód

Traverzovanie syntaktického stromu: „Visitor“

Uvedeným volaním sú okrem iného vytvorené:

- Rozhranie `GramatikaVisitor<T>` dediace od `ParseTreeVisitor<T>`
- Trieda `GramatikaBaseVisitor<T>` dediaca od `AbstractParseTreeVisitor<T>` a implementujúca `GramatikaVisitor<T>`
- `GramatikaBaseVisitor<T>` obsahuje východzie implementácie metód na návštevu koreňa podstromu (navštív všetky deti)
- Od `GramatikaBaseVisitor<T>` môže dediť trieda obsahujúca ozajstné implementácie týchto metód
- `T` je trieda, ktorej inštancie sú výstupmi metód na návštevu podstromov

Traverzovanie syntaktického stromu: „Visitor“

Uvedeným volaním sú okrem iného vytvorené:

- Rozhranie `GramatikaVisitor<T>` dediace od `ParseTreeVisitor<T>`
- Trieda `GramatikaBaseVisitor<T>` dediaca od `AbstractParseTreeVisitor<T>` a implementujúca `GramatikaVisitor<T>`
- `GramatikaBaseVisitor<T>` obsahuje východzie implementácie metód na návštevu koreňa podstromu (navštív všetky deti)
- Od `GramatikaBaseVisitor<T>` môže dediť trieda obsahujúca ozajstné implementácie týchto metód
- `T` je trieda, ktorej inštancie sú výstupmi metód na návštevu podstromov
- Od `AbstractParseTreeVisitor<T>` je zdedená metóda `T visit(ParseTree tree)`

Traverzovanie syntaktického stromu: „Visitor“

Uvedeným volaním sú okrem iného vytvorené:

- Rozhranie `GramatikaVisitor<T>` dediace od `ParseTreeVisitor<T>`
- Trieda `GramatikaBaseVisitor<T>` dediaca od `AbstractParseTreeVisitor<T>` a implementujúca `GramatikaVisitor<T>`
- `GramatikaBaseVisitor<T>` obsahuje východzie implementácie metód na návštevu koreňa podstromu (navštív všetky deti)
- Od `GramatikaBaseVisitor<T>` môže dediť trieda obsahujúca ozajstné implementácie týchto metód
- `T` je trieda, ktorej inštancie sú výstupmi metód na návštevu podstromov
- Od `AbstractParseTreeVisitor<T>` je zdedená metóda `T visit(ParseTree tree)`
- Táto metóda sa používa v implementáciách jednotlivých metód „vistora“ („kontexty“ dedia od `ParseTree`), ako aj v hlavnom programe na spustenie traverzovania

Ľavá a pravá asociatívnosť operátorov

- Automaticky identifikované operátory chápe ANTLR4 ako ľavo asociatívne

Ľavá a pravá asociatívnosť operátorov

- Automaticky identifikované operátory chápe ANTLR4 ako ľavo asociatívne
- Väčšinou je to v poriadku: napríklad $a - b - c = (a - b) - c$

Ľavá a pravá asociatívnosť operátorov

- Automaticky identifikované operátory chápe ANTLR4 ako ľavo asociatívne
- Väčšinou je to v poriadku: napríklad $a - b - c = (a - b) - c$
- Niektoré operátory sú pravo asociatívne: $a \hat{} b \hat{} c = a \hat{} (b \hat{} c)$

Ľavá a pravá asociatívnosť operátorov

- Automaticky identifikované operátory chápe ANTLR4 ako ľavo asociatívne
- Väčšinou je to v poriadku: napríklad $a - b - c = (a - b) - c$
- Niektoré operátory sú pravo asociatívne: $a \hat{} b \hat{} c = a \hat{} (b \hat{} c)$

ANTLR4 umožňuje špecifikovať asociativitu binárneho operátora na začiatku alternatívy:

Ľavá a pravá asociatívnosť operátorov

- Automaticky identifikované operátory chápe ANTLR4 ako ľavo asociatívne
- Väčšinou je to v poriadku: napríklad $a - b - c = (a - b) - c$
- Niektoré operátory sú pravo asociatívne: $a \hat{} b \hat{} c = a \hat{} (b \hat{} c)$

ANTLR4 umožňuje špecifikovať asociativitu binárneho operátora na začiatku alternatívy:

```
expr
      :    <assoc=right> expr op=POW expr    # BinOp
      |    expr op=(MUL|DIV) expr             # BinOp
      |    expr op=(ADD|SUB) expr             # BinOp
      |    LEFT expr RIGHT                    # Paren
      |    SUB LEFT expr RIGHT                # NegParen
      |    NUM                                # Number
      |    SUB NUM                            # NegNumber
      ;
```