Programski prevodioci 1

Apstraktna sintaksna stabla

Uvod

- Iako bi se mnogi zadaci kompajlera mogli izvršiti u jednoj fazi putem sintaksno-upravljanog prevođenja zasnovanog na atributivno-translacionim gramatikama, savremene softverske prakse obeshrabruju implementaciju većeg broja različitih funkcionalnosti u jednoj komponenti kao što je parser.
- Zadaci kao što su semantička analiza, konstrukcija tabele simbola, optimizacija programa, i generisanje koda zaslužuju zaseban tretman u kompajleru. Ako se svi ti zadaci smeste na jedno mesto (u specifikaciju semantičkih akcija gramatike) rezultirajući kompajler je teško razumeti, proširiti, i održavati.
- Stoga razmatramo dizajn i primenu strukture podataka, poznate kao **apstraktno sintaksno stablo** (AST), koji će služiti kao centralna struktura podataka za sve aktivnosti posle parsiranja. Cilj sintaksno-upravljanog prevođenja se tada svodi na izgradnju AST-a.
- Najčešće se u procesiranju ASTa koristi projektni šablon **Posetilac** (Visitor), koja obezbeđuje efikasan obilazak stabla i razdvajanje algoritma od strukture nad kojom operiše. Nove vrste procesiranja (kompajlerski prolazi) mogu se dodavati bez menjanja postojećih klasa koje definišu strukturu, dodavanjem novih posetilaca.

Tekući primer

- Da bismo ilustrovali osnovne koncepte (i način kako ćemo mi u mikrojavi koristiti AST) krenućemo od jednostavnog primera kalkulatora za osnovne celobrojne aritmetičke izraze sa operatorima +, -, *, /.
- Gramatika za ove izraze prilagođena parserima od dna ka vrhu je sledeća:

<Expr> → <Expr> <addop> <Term>
<Expr> → <Term>
<Term> → <Term> <mulop> <Factor>
<Term> → <Factor>
<Factor> → NUMBER
<addop> → PLUS | MINUS
<mulop> → STAR | DIVIDE

Tekući primer

- Za implementaciju parsera prepoznavača alatima jflex i cup dovoljna su tri fajla:
- calc.lex je specifikacija leksičkog analizatora

```
1
2 package parser;
3 import java_cup.runtime.Symbol;
4 %%
5 %cup
6 %%
7 "+" { return new Symbol(sym.PLUS); }
8 "-" { return new Symbol(sym.MINUS); }
9 "*" { return new Symbol(sym.STAR); }
10 "/" { return new Symbol(sym.DIVIDE); }
11 [0-9]+ { return new Symbol(sym.NUMBER); }
12 [ \t\r\n\f] { /* ignore white space. */ }
13 . { System.err.println("Illegal character: "+yytext()); }
```

Tekući primer

- calc.cup je specifikacija sintaksnog analizatora

```
 1 package parser;
 2 terminal NUMBER;
 3 terminal PLUS, MINUS, STAR, DIVIDE;
 4 nonterminal expr, term, factor;
 5 nonterminal addop, mulop;
 6 expr ::= term
 7 | expr addop term;
 8 term ::= factor
 9 | term mulop factor;
10 factor ::= NUMBER;
11 addop ::= PLUS   | MINUS ;
12 mulop ::= STAR   | DIVIDE ;
```

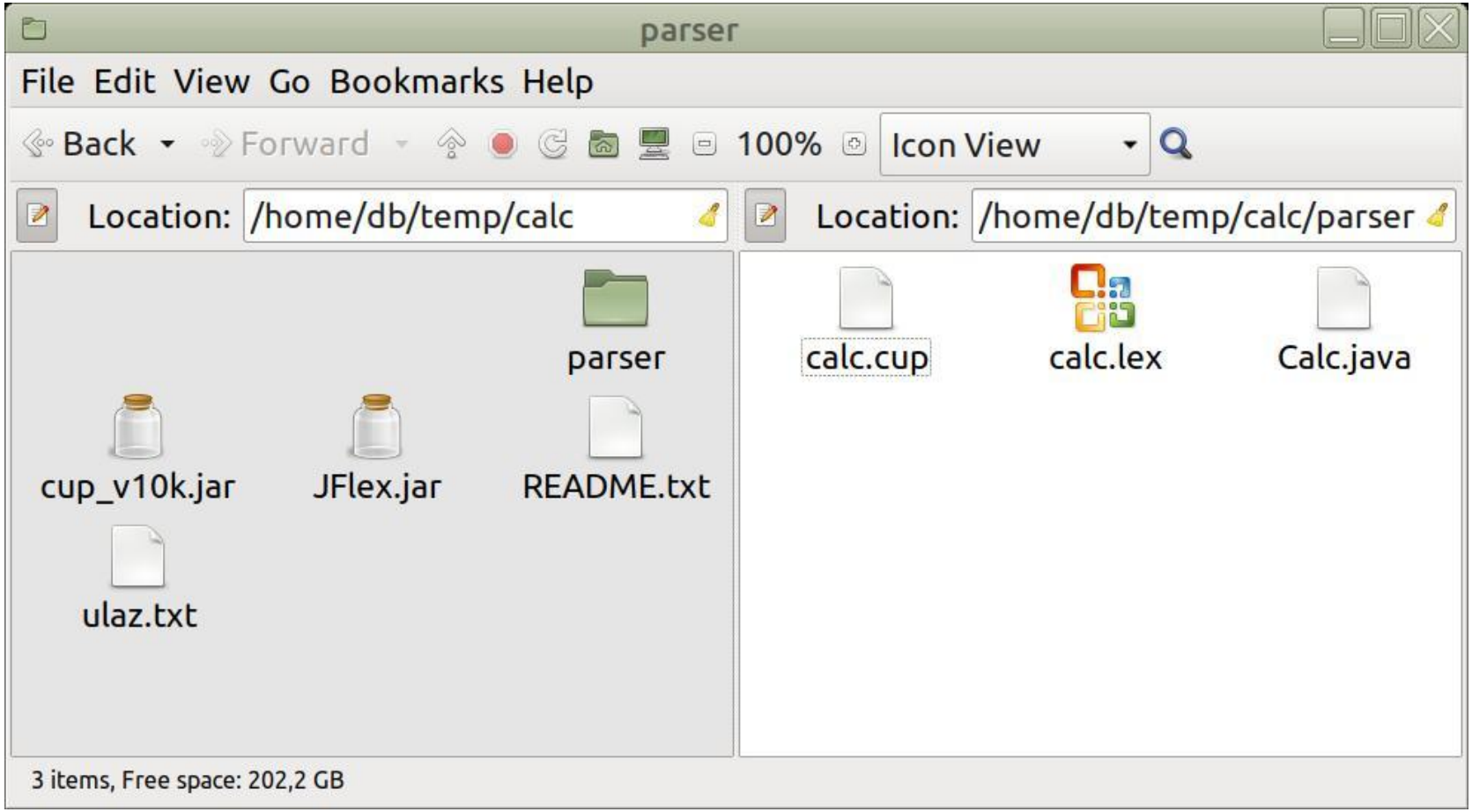
Tekući primer

- Calc.java sadrži glavni program koji instancira skener i parser, dostavlja im ulaz i pokreće parsiranje

```
 1
 2 package parser;
 3
 4 import java_cup.runtime.*;
 5 import java.io.*;
 6
 7 class Calc {
 8     public static void main(String args[]) throws Exception {
 9         FileReader r = new FileReader(args[0]);
10         Yylex skener = new Yylex(r);
11         parser p = new parser(skener);
12         Symbol s = p.debug_parse();
13     }
14 }
```

Tekući primer

- Struktura fajlova na disku



Tekući primer

- Prevođenje i pokretanje primera iz terminala (na Linuxu, za Windows treba zameniti / sa \ i : sa ;)

- java -jar JFlex.jar -d parser parser/calc.lex
  kreira izvorni fajl skenera u parser/Yylex.java

- java -jar cup_v10k.jar -destdir parser parser/calc.cup
  kreira izvorne fajlove parsera parser.java i definicije tokena sym.java (ovaj drugi fajl koristi i skener)

```
------- CUP v0.10k Parser Generation Summary -------
  0 errors and 0 warnings
  7 terminals, 6 non-terminals, and 10 productions declared,
  producing 14 unique parse states.
  0 terminals declared but not used.
  0 non-terminals declared but not used.
  0 productions never reduced.
  0 conflicts detected (0 expected).
  Code written to "parser.java", and "sym.java".
---------------------------------------------------- (v0.10k)
```

Tekući primer

- javac -cp .:cup_v10k.jar parser/Calc.java
  prevodi sve .java fajlove u .class fajlove u folderu parser (ne bi trebalo ništa da prijavi na izlazu ako je sve u redu)

- java -cp .:cup_v10k.jar parser.Calc ulaz.txt
  pokretanje programa, parsira se ulaz.txt. Pošto je pozvan debug_parse, na izlazu će se prikazati trace rada parsera za 3+5

```
# Initializing parser
# Current Symbol is #2
# Shift under term #2 to state #3
# Current token is #3
# Reduce with prod #5 [NT=3, SZ=1]
# Reduce rule: top state 0, lhs sym 3 -> state 1
# Goto state #1
# Reduce with prod #3 [NT=2, SZ=1]
# Reduce rule: top state 0, lhs sym 2 -> state 2
# Goto state #2
# Reduce with prod #0 [NT=1, SZ=1]
# Reduce rule: top state 0, lhs sym 1 -> state 4
# Goto state #4
# Shift under term #3 to state #6
# Current token is #2
# Reduce with prod #6 [NT=4, SZ=1]
# Reduce rule: top state 4, lhs sym 4 -> state 5
# Goto state #5
# Shift under term #2 to state #3
# Current token is #0
# Reduce with prod #5 [NT=3, SZ=1]
# Reduce rule: top state 5, lhs sym 3 -> state 1
# Goto state #1
# Reduce with prod #3 [NT=2, SZ=1]
# Reduce rule: top state 5, lhs sym 2 -> state 9
# Goto state #9
# Reduce with prod #2 [NT=1, SZ=3]
# Reduce rule: top state 0, lhs sym 1 -> state 4
# Goto state #4
# Shift under term #0 to state #8
# Current token is #0
# Reduce with prod #1 [NT=0, SZ=2]
# Reduce rule: top state 0, lhs sym 0 -> state -1
# Goto state #-1
db@zbook:~/temp/calc$
```

Proširenje parsera semantičkom obradom

- Želimo da kalkulator računa vrednost izraza. Ova obrada se može specificirati sledećom atributivno-translacionom gramatikom:

1. $\langle Expr \rangle_r \rightarrow \langle Expr \rangle_e \langle addop \rangle_{op} \langle Term \rangle_t$
   $r \leftarrow$ (op==‘add’) ? e+t : e-t
2. $\langle Expr \rangle_r \rightarrow \langle Term \rangle_t$
   $r \leftarrow t$
3. $\langle Term \rangle_r \rightarrow \langle Term \rangle_t \langle mulop \rangle_{op} \langle Factor \rangle_f$
   $r \leftarrow$ (op==‘mul’) ? t*f : t/f
4. $\langle Term \rangle_r \rightarrow \langle Factor \rangle_f$
   $r \leftarrow f$
5. $\langle Factor \rangle_r \rightarrow NUMBER_{val}$
   $r \leftarrow val$
6. $\langle addop \rangle_r \rightarrow PLUS$ $r \leftarrow$ ‘add’
7. $\langle addop \rangle_r \rightarrow MINUS$ $r \leftarrow$ ‘sub’
8. $\langle mulop \rangle_r \rightarrow STAR$ $r \leftarrow$ ‘mul’
9. $\langle mulop \rangle_r \rightarrow DIVIDE$ $r \leftarrow$ ‘div’

Tekući primer sa atributivnom translacijom

- calc.lex, izmene u specifikaciji leksičkog analizatora
- Terminal NUMBER sada ima atribut koji predstavlja vrednost konstante

```
package atributi;
import java_cup.runtime.Symbol;
%%
%cup
%%
"+" { return new Symbol(sym.PLUS); }
"-" { return new Symbol(sym.MINUS); }
"*" { return new Symbol(sym.STAR); }
"/" { return new Symbol(sym.DIVIDE); }
[0-9]+ { return new Symbol(sym.NUMBER, new Integer(yytext())); }
[ \t\r\n\f] { /* ignore white space. */ }
. { System.err.println("Illegal character: "+yytext()); }
```

Tekući primer sa atributivnom translacijom

- calc.cup, izmene u specifikaciji sintaksnog analizatora
- Neterminali su dobili svoje atribute (tip podataka iza ključne reči nonterminal), takođe je i terminal NUMBER dobio svoj atribut.
- Atribut neterminala ne može biti ugrađeni tip int zbog ograničenja alata mora klasa tj. Integer
- Atributivna pravila su ugrađena u semantičke akcije smena {: :}
- Na desnim stranama smena imenovani su atributi koji se koriste u akcijama
- Sve ovo čini gramatičku specifikaciju nešto manje preglednom

```
package atributi;
terminal int NUMBER;
terminal PLUS, MINUS, STAR, DIVIDE;
nonterminal Integer expr, term, factor;
nonterminal String addop, mulop;
expr ::= term:t {: RESULT = t; :}
| expr:e addop:op term:t {: RESULT = new Integer ((op.equals("add"))? e.intValue()+t.intValue() : e.intValue()-t.intValue()); :};
term ::= factor:f {: RESULT = f; :}
| term:t mulop:op factor:f {: RESULT = new Integer((op.equals("mul"))? t.intValue()*f.intValue() : t.intValue()/f.intValue()); :};
factor ::= NUMBER:val {: RESULT = new Integer(val); :};
addop ::= PLUS {: RESULT="add"; :} | MINUS {: RESULT="sub"; :};
mulop ::= STAR {: RESULT="mul"; :} | DIVIDE {: RESULT="div"; :};
```

Tekući primer sa atributivnom translacijom

- Calc.java, izmene u glavnom programu
- Parse funkcija vraća atribut startnog simbola (ali preko pokazivača na interfej Symbol)
- Konverzijom u pravi tip uzima se celobrojna vrednost i štampa naizlazu

```
package atributi;

import java_cup.runtime.*;
import java.io.*;

class Calc {
    public static void main(String args[]) throws Exception {
        FileReader r = new FileReader(args[0]);
        Yylex skener = new Yylex(r);
        parser p = new parser(skener);
        Symbol s = p.parse();
        Integer result = (Integer)(s.value);
        System.out.println(result.intValue());
    }
}
```

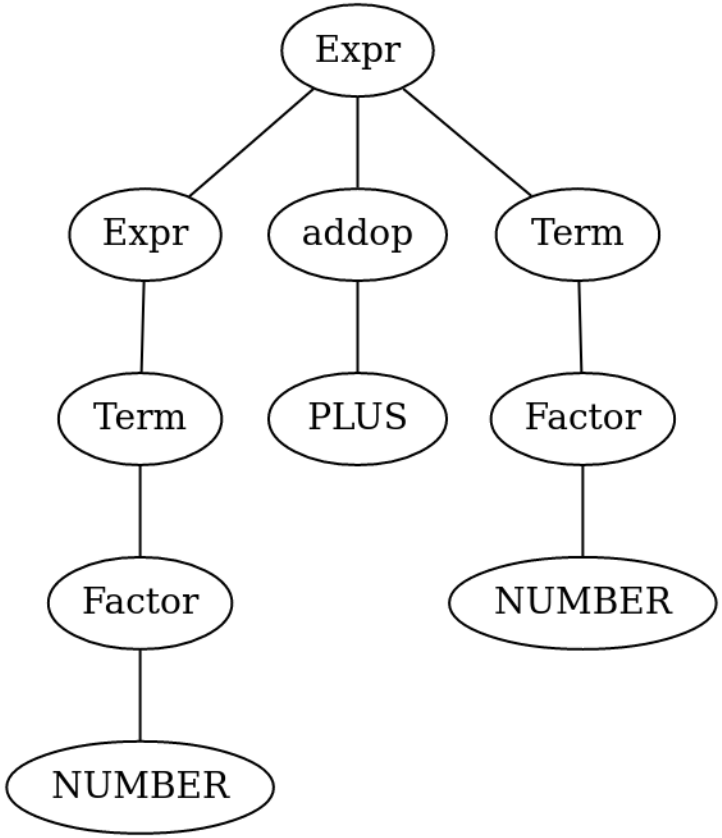
Tekući primer sa atributivnom translacijom

- Prevođenje i pokretanje kao ranije, samo što je ime paketa atributi umesto parser

- java -jar JFlex.jar -d atributi atributi/calc.lex
- java -jar cup_v10k.jar -destdir atributi atributi/calc.cup
- javac -cp .:cup_v10k.jar atributi/Calc.java
- java -cp .:cup_v10k.jar atributi.Calc ulaz.txt
  Sada ispisuje na izlazu rezultat 8 za ulaz 3+5

Konkretno i apstraktno sintaksno stablo

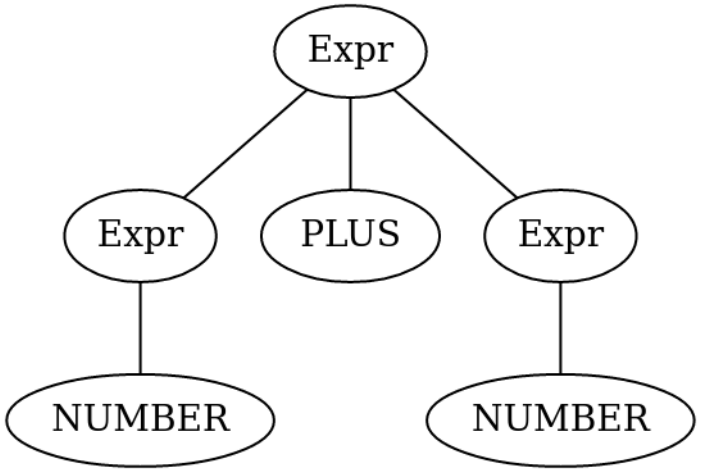
- Kao što znamo od ranije, konkretno sintaksno stablo striktno odgovara gramatici i izvođenju neke konkrente ulazne sekvence. Na primer, za našu gramatiku kalkulatora i ulaznu sekvencu 3+5 imamo sledeće konkretno sintaksno stablo (bez atributa)



Konkretno i apstraktno sintaksno stablo

- Izraze bismo mogli opisati znatno jednostavnijom gramatikom koja opisuje njihovu suštinu: operandi (vrednosti) i operacije.

<Expr> → <Expr> PLUS <Expr>
<Expr> → <Expr> MINUS <Expr>
<Expr> → <Expr> STAR <Expr>
<Expr> → <Expr> DIVIDE <Expr>
<Expr> → NUMBER



- Izrazu 3+5 iz tekućeg primera odgovara prikazano sintaksno stablo, koje možemo nazvati apstraktnim sintaksnim stablom za izraze.
- Pogodnost ovog stabla za dalju obradu je što su izostavljeni nebitni detalji i u opštem slučaju znatno je manjeg obima. Dakle kod za semantičku obradu može biti koncizniji i efikasniji.

Konkretno i apstraktno sintaksno stablo

- Problem sa uprošćenom gramatikom je njena dvosmislenost i činjenica da ne specificira asocijativnost niti prioritete operatora
- Međutim, ako za parsiranje izraza koristimo početnu gramatiku i parser, a zatim na osnovu konkretnog stabla izvođenja za sekvencu konstruišemo uprošćeno (apstraktno) sintaksno stablo, rešili smo problem dobijanja apstraktnog stabla.
- Apstraktno stablo možemo dobiti ako odgovarajuće akcije za konstrukciju i povezivanje čvorova stabla ubacimo kao semantičke akcije u početni parser za konkretnu sintaksu.
- Ostaje međutim problem u opštem slučaju, mapiranja konkretne sintakse (programskog) jezika na njegovu apstraktnu sintaksu. Moraju se odvojeno definisati konkretna i apstraktna sintaksa i manuelno konstruisati AST iz konkretnog stabla.

Konkretno i apstraktno sintaksno stablo

- Alternativni pristup je da se iskoristi početna konkretna sintaksa jezika i automatski izgeneriše AST, koje u tom slučaju nema mnogo apstrakcije (mogu istina da se "odseku" delovi konkretnog stabla izvođenja od izabranog čvora na niže ako nemaju semantički značaj), ali se uz minimalne intervencije na polaznoj gramatici automatski generišu sve potrebne strukture podataka za dalje kompajlerske obrade.
- Ovaj pristup primenjen je proširivanjem standarnog cup alata, osmišljen u sklopu ovog kursa i realizovan kroz diplomski rad kolege Dušana Stankovića.
- Na primeru kalkulatora izložićemo osnovne koncepte rada proširenog cup alata

Tekući primer sa AST

- calc.lex je isti kao kod varijante sa atributivnom translacijom
- calc.cup, izmene u odnosu na varijantu čist parser

```
package ast;
terminal int NUMBER;
terminal PLUS, MINUS, STAR, DIVIDE;
nonterminal Value expr, term, factor;
nonterminal String addop, mulop;
expr ::= (SingleExpr) term
| (Addition) expr addop term;
term ::= (SingleTerm) factor
| (Multiplication) term mulop factor;
factor ::= (Constant) NUMBER:val;
addop ::= PLUS {: RESULT="add"; :} | MINUS {: RESULT="sub"; :};
mulop ::= STAR {: RESULT="mul"; :} | DIVIDE {: RESULT="div"; :};
```

- Value.java je dodata klasa da bi se realizovalo čuvanje vrednosti za čvorove Expr, Term, Factor

```
1 package ast;
2
3 class Value {
4     public int v;
5 }
```

Tekući primer sa AST

- Calc.java, izmene u odnosu na varijantu čist parser

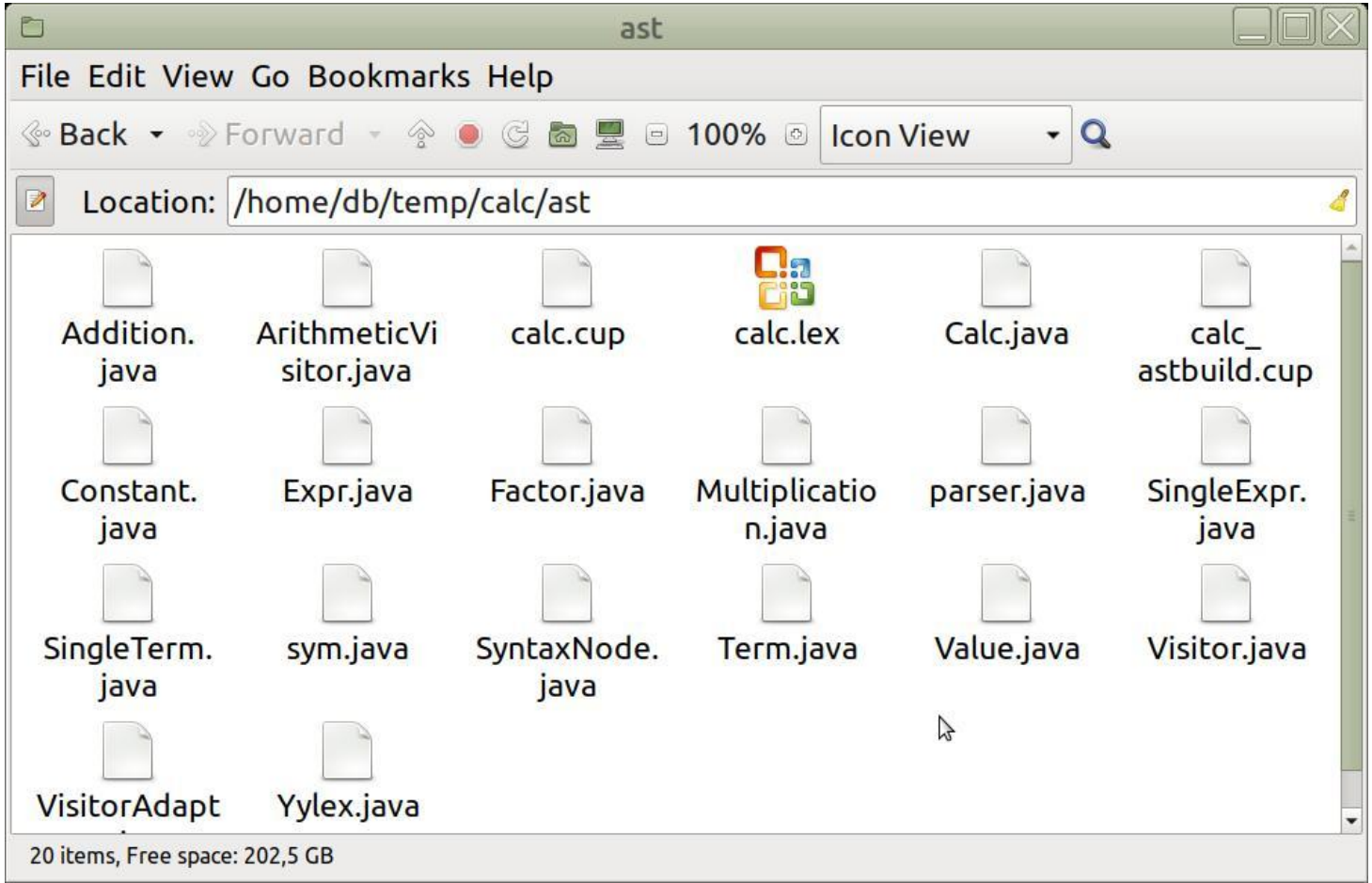
```
package ast;

import java_cup.runtime.*;
import java.io.*;

class Calc {
        public static void main(String args[]) throws Exception
                FileReader r = new FileReader(args[0]);
                Yylex skener = new Yylex(r);
                parser p = new parser(skener);
                Symbol s = p.parse();
                Expr e = (Expr)(s.value);
                ArithmeticVisitor v = new ArithmeticVisitor();
                e.traverseBottomUp(v);
                System.out.println(e.value.v);
                System.out.println(e.toString());
        }
}
```

Tekući primer sa AST

- Dodat je još jedan novi fajl ArithmeticVisitor.java, koji ćemo malo kasnije proučiti
- Generisanje izvornih fajlova skenera, parsera i podrške za AST:

  java -jar JFlex.jar -d ast ast/calc.lex

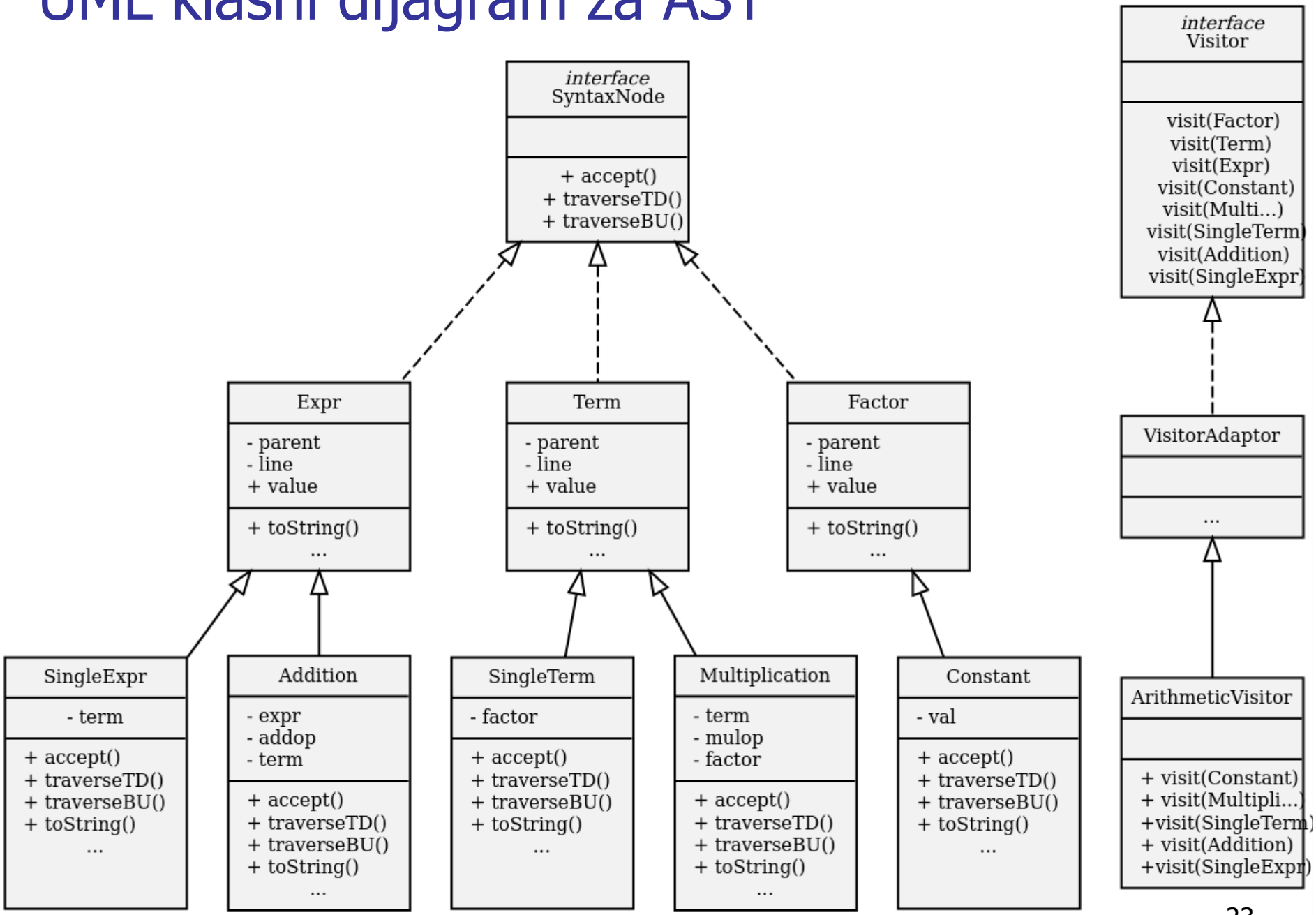
  java -jar cup_v10k.jar -destdir ast -ast ast -buildtree ast/calc.cup



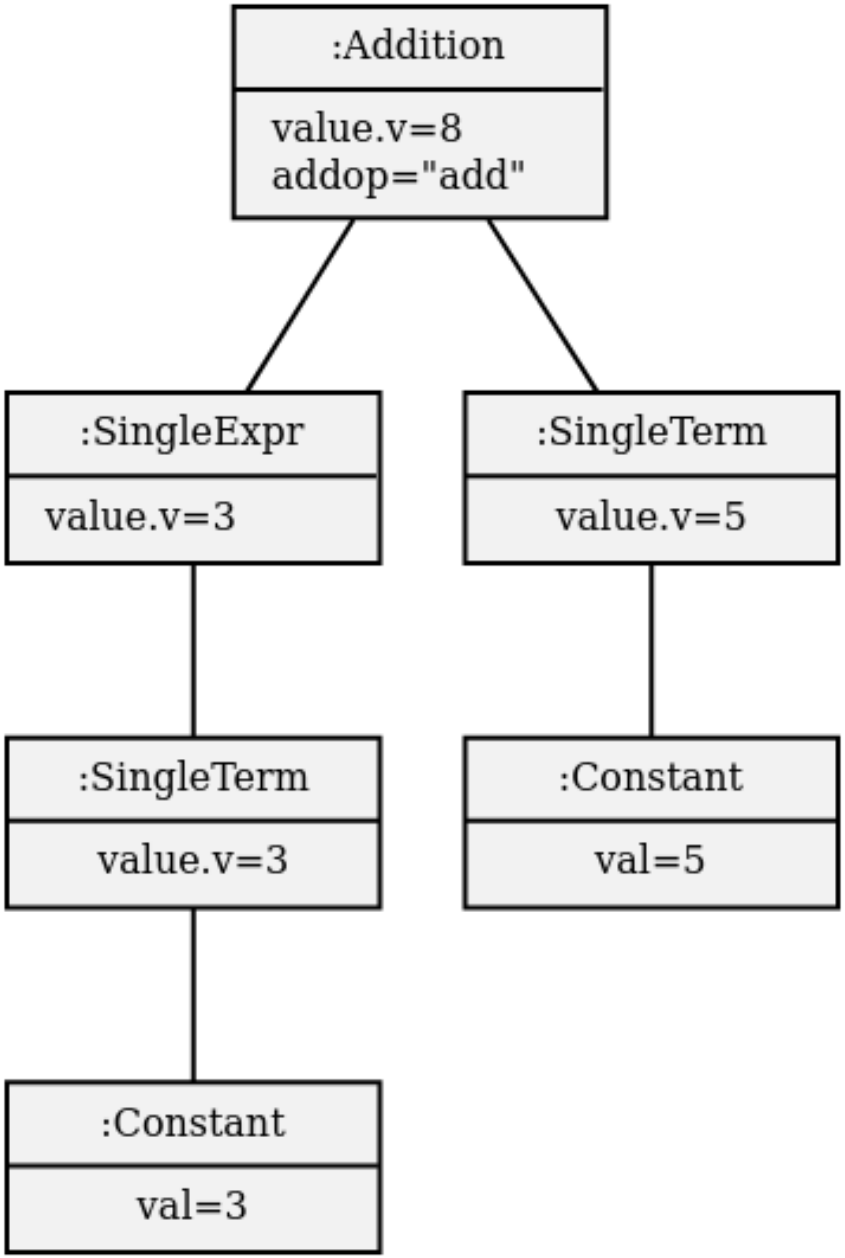
Tekući primer sa AST

- Pojavilo se puno novih .java fajlova:
- parser.java, sym.java i Yylex.java su kao i ranije, skener i parser i definicija tokena
- calc_astbuild.cup je fajl koji interno koristi prošireni cup alat da izgeneriše parser sa automatskim konstruisanjem AST stabla
- Fajlovi SyntaxNode.java, Visitor.java, VisitorAdapter.java, Expr.java, Term.java, Factor.java, Addition.java, Multiplication.java, SingleExpr.java, SingleTerm.java, Constant.java predstavljaju podršku za AST sa obilaskom čvorova po Visitor šablonu.
- Svi ovi fajlovi su automatski generisani i ne sme se u njima ništa ručno menjati, jer će te izmene biti uklonjene pri ponovnom pokretanju cup alata

UML klasni dijagram za AST



Objektni dijagram AST za ulaz 3+5



Tekući primer sa AST

- ArithmeticVisitor je poslednja klasa koja je ručno napisana a koju još nismo proučili. Ovde se nalazi kompletna semantička obrada, u ovom primeru to je računanje vrednost ulaznog izraza reprezenovanog AST stablom

```
1  package ast;
2
3  public class ArithmeticVisitor extends VisitorAdaptor {
4
5      public void visit(Constant constant) {
6          constant.value = new Value();
7          constant.value.v = constant.getVal();
8      }
9
10     public void visit(Multiplication multiplication) {
11         multiplication.value = new Value();
12         int op1=multiplication.getTerm().value.v;
13         int op2=multiplication.getFactor().value.v;
14         if(multiplication.getMulop().equals("mul")) {
15             multiplication.value.v = op1*op2;
16         } else {
17             multiplication.value.v = op1/op2;
18         }
19     }
20
21     public void visit(SingleTerm SingleTerm) {
22         SingleTerm.value = SingleTerm.getFactor().value;
23     }
```

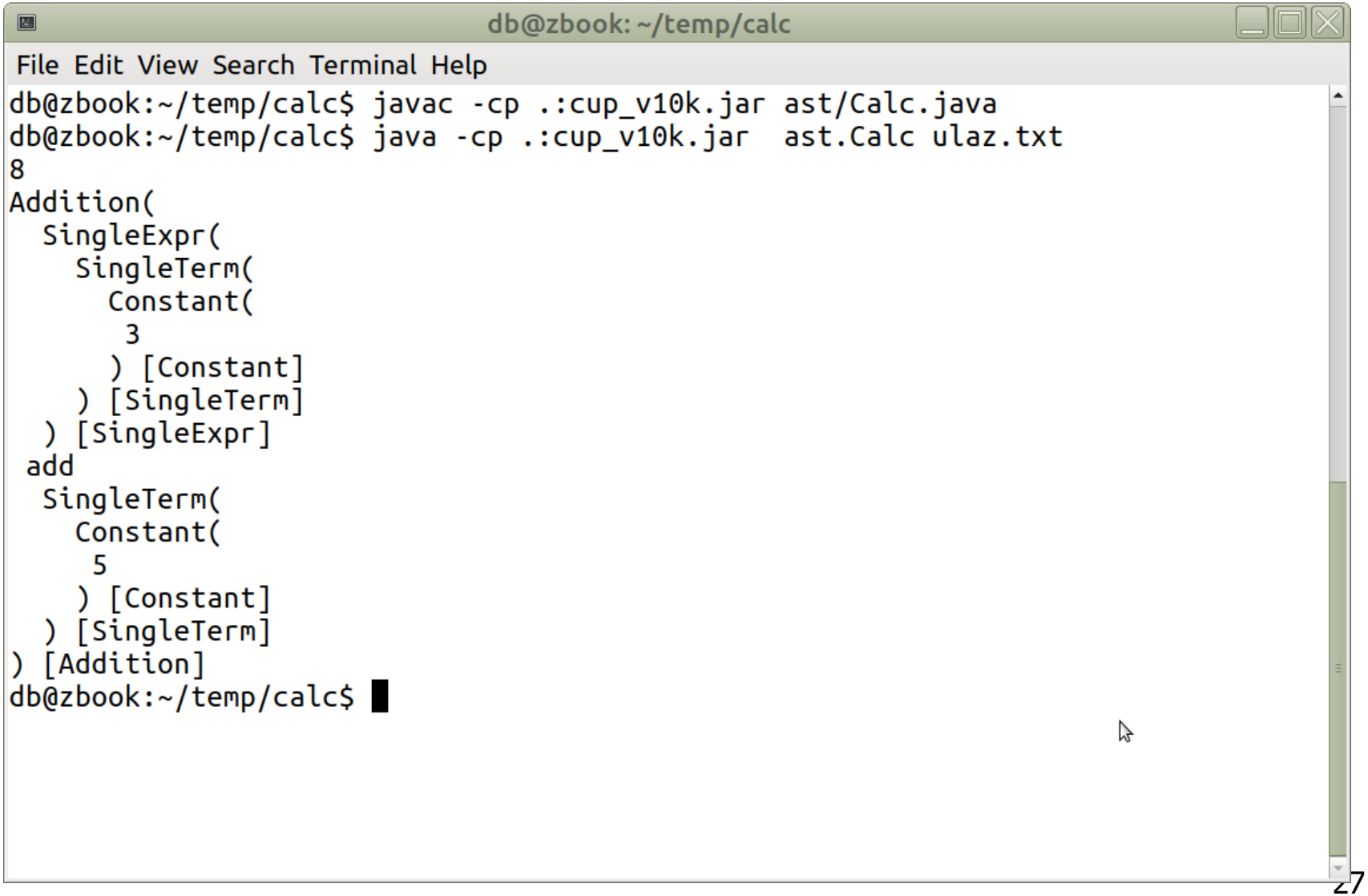
Tekući primer sa AST

- ArithmeticVisitor, nastavak.

```
24
25
26      public void visit(Addition addition) {
27          addition.value = new Value();
28          int op1=addition.getExpr().value.v;
29          int op2=addition.getTerm().value.v;
30          if(addition.getAddop().equals("add")) {
31              addition.value.v = op1+op2;
32          } else {
33              addition.value.v = op1-op2;
34          }
35      }
36
37      public void visit(SingleExpr SingleExpr) {
38          SingleExpr.value = SingleExpr.getTerm().value;
39      }
40
41  }
```

Tekući primer sa AST

- Prevođenje i pokretanje programa. Na izlazu se dobija tekstualni opis AST stabla i finalna vrednost izraza.



Završne napomene

- Postavlja se pitanje, da li je varijanta sa AST, koja deluje znatno složenije nego sa atributima i akcijama u gramatici, “vredna truda”?
- Za ovaj prost primer možda i nije, ali kod ozbiljnih primena, svaki semantički prolaz odvojio bi se pisanjem posebnog visitora i .cup fajl ne bi se ni na koji način opterećivao.
- Znatno složeniji primer sa korišćenjem AST, praktično kompletan kompajler sa sintaksnom, semantičkom analizom i generisanjem koda za podskup jezika mikrojava nalazi se u okviru vežbi na sajtu predmeta (mini domaći).