Programski prevodioci (vežbe)

Generatori sintaksnih analizatora

Generatori sintaksnih analizatora

- Yacc – prvi generator sintaksnih analizatora nastao u Bell laboratorijama 1975. (zajedno sa Lex-om) kao deo operativnog sistema UNIX. Generiše sintaksni analizator (parser) na programskom jeziku C.
- CUP (Construct Useful Parser) generiše LR parser u Javi (i sam je pisan u javi). Ovo je besplatan “open-source” alat.

CUP linkovi

- CUP alat http://www2.cs.tum.edu/projects/cup/
- CUP Eclipse plugin http://www2.cs.tum.edu/projects/cup/eclipse.php

CUP arhitektura

- Fiksni delovi aplikacije (nezavisni od jezika):
  - Klasa **Symbol** koja služi za predstavljanje podataka o svim simbolima gramatike.
  - Interfejs **Scanner**. Klasa koja služi za leksičku analizu mora da impelmentira ovaj interfejs. U interfejsu je definisana funkcija za leksičku analizu (**next_token()**) koja vraća rezultat tipa Symbol.
  - Klasa **lr_parser** koja sadrži funkciju za LR sintaksnu analizu (funkciju **parse()**) koja se bazira na korišćenju sintaksnih tabela koje su zavisne od jezika pa se generišu u izvedenoj klasi. Rezultat funkcije parse() je tipa Symbol i ako je analiza uspešno završena, to je startni simbol gramatike.

CUP arhitektura

- Generisani deo aplikacije (zavisan od jezika):
  - Klasa **sym** koja sadrži definicije tipova terminalnih simbola (celobrojne identifikatore tokena).
  - Klasa **parser** (čije ime može biti i promenjeno) koja je izvedena iz klase lr_parser i koja sadrži funkcije za inicjalizaciju potrebnih tabela:
    - production_tab – sadrži identifikacioni broj simbola sa leve strane smene i dužinu desne strane.
    - action_tab – sadrži deo za akcije LR sintaksne tabele.
    - reduce_tab – sadrži deo za prelaze nakon redukcija u LR st.
  - Privatna klasa **CUP$action** koja sadrži definicije akcija (akcije koje korisnik definiše u cup specifikaciji). Najbitnija funkcija ove klase je **CUP$do_action()**.

Klasa Symbol

```
public class Symbol {
 /** The symbol number of the terminal or non terminal being represented */
  public int sym;

    /** The parse state to be recorded on the parse stack with this symbol.
   *  This field shouldn't be modified except by the parser. */
  public int parse_state;

  /** This allows us to catch some errors caused by scanners recycling symbols.
  * For the use of the parser only. */
  boolean used_by_parser = false;

/*********************************
  The data passed to parser
 *********************************/
  public int left, right;
  public Object value;
}
```

Implementacija leksičkog analizatora

Leksički analizator, koji parser generisan pomoću cup-a koristi, mora da zadovolji sledeće ulsove:

- Implementira interfejs Scanner,
- Funkcija za leksičku analizu je next_token(),
- Rezultat leksičke analize je objekat klase Symbol.

Implementacija leksičkog analizatora

Leksički analizator može biti implementiran u javi bez korišćenja alata, a može biti i generisan pomoću jflex-a. Ukoliko se leksički analizator generiše pomoću jflex-a, u flex specifikaciji obavezno je navođenje sledećih direktiva:

```
%implements  java_cup.runtime.Scanner
%function  next_token
%type  java_cup.runtime.Symbol
```

Umesto navedenih dovoljno je navesti direktivu:

```
%cup
```

što znači da generisani skener treba da bude “cup kompatibilan”.

CUP specifikacija

CUP specifikacija sadrži 3 dela koji se obavezno pišu u navedenom redosledu:

- Import sekcija,
- Definicije terminalnih i neterminalnih simbola gramatike,
- Opis gramatike.

Import sekcija

Import sekcija sadrži delove koda koji se direktno ugrađuju u generisani kod bez provere njihove sintaksne ispravnosti. Ovaj deo sadrži:

- Import i package naredbe koje se prepisuju na početak fajla u koji se upisuje klasa parser.
- Definicije novih članova (atributa i funkcija) klase parser.

  parser code {: ... :};

  U ovom delu se obično predefinišu i funkcije za prikaz poruka o greškama.
- Definicije novih članova (atributa i funkcija) klase CUP$action.

  action code {: ... :};
- Deo koda koji se izvršava pre izdvajanja prve reči iz ulaznog fajla (tj. pre prvog poziva scanner-a).

  init with {: ... :};
- Deo koda koji definiše kako parser poziva leksički analizator, tj. kako traži izdvajanje sledeće reči.

  scan with {: ... :};

Definicije terminalnih i neterminalnih simbola

- Definisanje terminalnih simbola

  terminal [<ImeKlase>] <ImeSimbola1>, <ImeSimbola2>, … ;

  <ImeKlase> je tip člana value u objektu Symbol koji odgovara datom terminalnom simbolu.

- Definisanje neterminalnih simbola

  non terminal [<*ImeKlase*>] <*ImeSimbola1*>, <*ImeSimbola2*>, … ;

Definicije asocijativnosti i prioriteta operatora:

Operatori mogu biti:

- Levoasocijativni,
- Desnoasocijativni,
- Neasocijativni.

- Definicije asocijativnosti:

  precedence left <*ImeSimbola1*>, <*ImeSimbola2*>, … ;

  precedence right <*ImeSimbola1*>, <*ImeSimbola2*>, … ;

  precedence nonassoc <*ImeSimbola1*>, <*ImeSimbola2*>, … ;

- Definicije prioriteta:

  Prioritet operatora je određen redosledom navođenja asocijativnosti. Asocijativnosti operatora se navode počev od operatora najnižeg prioriteta.

Primer1: Definisanja prioriteta i asocijativnosti operatora

Neka je gramatika za definisanje izraza data sledećim skupom smena:

Izraz → Izraz + Izraz | Izraz - Izraz

| Izraz * Izraz | Izraz / Izraz | - Izraz | **CONST**

U ovakvim situacijama (kada je gramatika nejednoznačna), potrebno je definisati prioritete i asocijativnosti operatora.

```
precedence left PLUS, MINUS;
precedence left PUTA, PODELJENO;
precedence right UMINUS;
```

Opis gramatike

- Na početku ovog dela se može (ne mora) definisati startni simbol gramatike deklaracijom:

  start with <*ImeSimbola*>;

  Ukoliko je ova deklaracija izostavljena, podrazumeva se da je startni simbol simbol koji se nalazi na levoj strani prve navedene smene.
- Za svaku smenu gramatike u cup specifikaciji se navodi po jedno pravilo oblika:

  <*LevaStrana*> ::= **<*DesnaStrana*>** [{: **<*Akcija*>** :}];

  Akcija predstavlja niz Java naredbi koje se izvršavaju u trenutku redukcije po navedenoj smeni.

Kraći zapis više smena koje na levoj strani imaju isti neterminalni simbol

Umesto:

```
<SymbolX> ::= <DesnaStrana1> [{: <Akcija1> :}];
<SymbolX> ::= <DesnaStrana2> [{: <Akcija2> :}];
...
 <SymbolX> ::= <DesnaStranaN> [{: <AkcijaN> :}];
```

Možemo pisati:

```
<SymbolX> ::= <DesnaStrana1> [{: <Akcija1> :}]
            |  <DesnaStrana2> [{: <Akcija2> :}]
      ...
            |  <DesnaStranaN> [{: <AkcijaN> :}]
            ;
```

Primer 2: Cup opis gramatike iz primera 1

```
Izraz ::= Izraz PLUS Izraz
            {: System.out.println("Prepoznat zbir"); :}
        | Izraz MINUS Izraz
            {: System.out.println("Prepoznata razlika"); :}
        | Izraz PUTA Izraz
            {: System.out.println("Prepoznat proizvod"); :}
        | Izraz PODELJENO Izraz
            {: System.out.println("Prepoznat količnik"); :}
        | MINUS Izraz
            {: System.out.println("Prepoznata promena znaka); :}
            %prec UMINUS          // "kontekstni prioritet"
        | CONST
            {: System.out.println("Prepoznata konstanta"); :}
        ;
```

Korišćenje vrednosti simbola koji se pojavljuju u smeni

Simboli na desnoj strani smene mogu biti označeni (imenovani). Ime simbola mora da bude jedinstveno za datu smenu.

- Navođenje imena simbola:

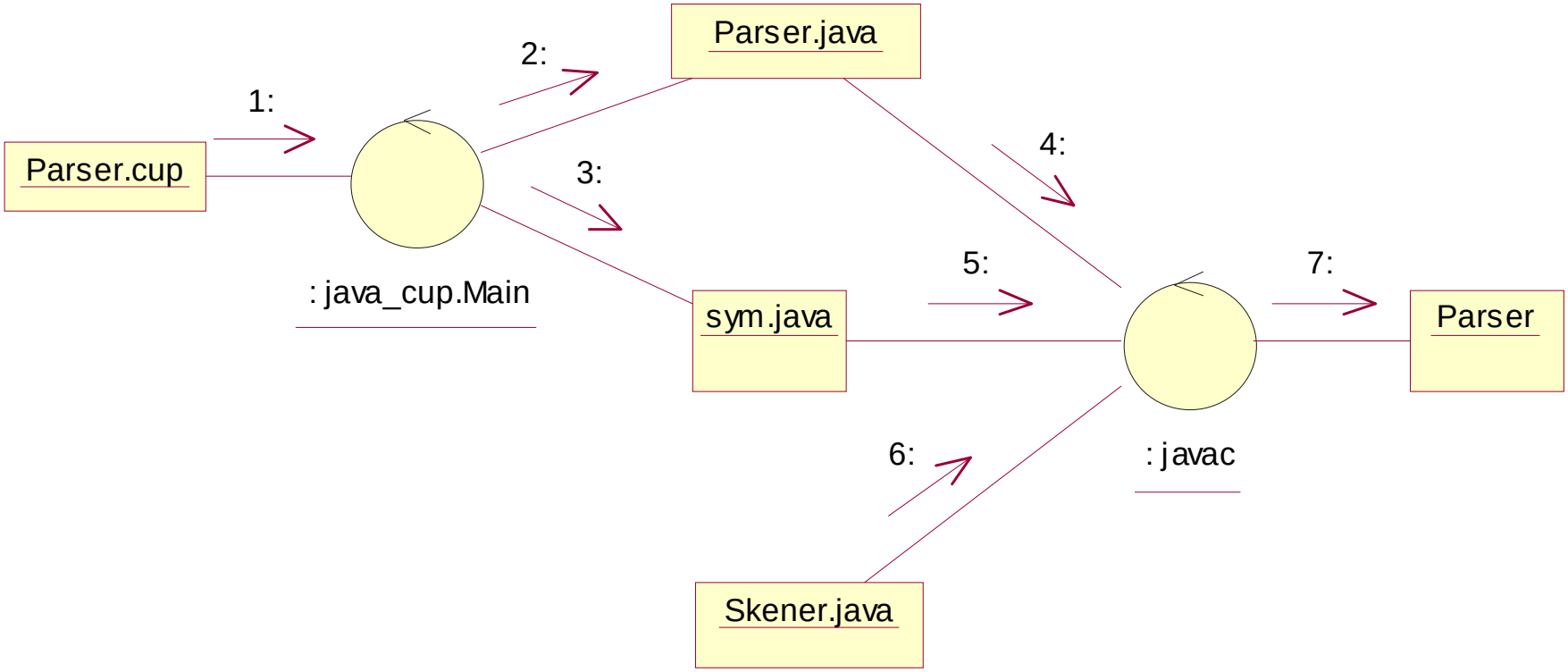
  <Simbol> : <Ime>

- Primer imenovanih simbola:

  Izraz ::= Izraz : i1 PLUS Izraz : i2

Imena simbola se u akciji koriste za pristup value atributu odgovarajućeg simbola. Value atributu simbola koji se nalazi na levoj strani smene pristupa se korišćenjem imena RESULT.

Od specifikacije do parsera



Generisanje parsera pomoću java_cup-a

Iz komandne linije:

java java_cup.Main [opcije] ImeSpec.cup

Najčešće korišćene opcije:

-parser <Ime> (menja ime klase parser u dato ime),

-symbols <Ime> (menja ime klase sym u dato ime),

-package <Ime> (specificira ime paketa u koji će biti smeštene klase parser i sym, po default-u one su u paketu java_cup.runtime),

-nonterminals (specificira da i identifikatori neterminalnih simbola budu zapisani u klasi sym).

Korišćenje generisanog koda

- Kreiranje objekta klase parser:
  - Konstruktoru klase parser se prosleđuje objekat tipa Scanner:

```
import java_cup.runtime.*;
...
FileReader = new FileReader( “Filename” );
parser = new parser( new ScanClass( input ) );
```

- Poziv funkcije za parsiranje:

```
Symbol syntax_tree = parser.parse();
```

Rad parsera

- Ukoliko je analizirani kod korektno zapisan generisani parser će izvršiti redukciju tog koda na startni simbol gramatike i vratiti taj simbol kao rezultat.
- Ukoliko se u kodu pojavi bilo kakva sintaksna greska, poziva se funkcija:

  void syntax_error(Symbol currentToken),

  a zatim se pokušava da se izvrši oporavak od greške. Ukoliko oporavak ne uspe, poziva se funkcija:

  void unrecovered_syntax_error(Symbol currentToken).

Funkcije koje se pozivaju u trenutku otkrivanja sintaksnih grešaka

- public void report_error(String message, Object info)
  - Ova funkcija služi za prikaz poruke o sintaksnoj grešci. U postojećoj implementaciji ona samo prikazuje datu poruku, dok se drugi argument prosto ignoriše. Drugi argument služi ako u predefinisanoj funkciji želimo da poruku o grešci "obogatimo" dodatnim informacijama.
- public void report_fatal_error(String message, Object info)
  - Ovu metodu treba pozvati za prikaz poruke o grešci kada oporavak nije moguć. U postojećoj implementaciji ona poziva metodu report_error(), poziva funkciju za prekid parsiranja done_parsing() i na kraju prijavljuje izuzetak.
- public void syntax_error(Symbol cur_token)
  - Ova metoda se poziva uvek kada se otkrije sintaksna greška. U postojećoj implementaciji ona sadrži samo poziv funkcije:

    report_error("Syntax error", null);.
- public void unrecovered_syntax_error(Symbol cur_token)
  - Ova metoda se poziva kada ne uspe oporavak od greške. U postojećoj implementaciji ona sadrži samo poziv:

    report_fatal_error("Couldn't repair and continue parse", null);.
- Sve ove funkcije se po pravilu predefinišu u delu "parser code".

Primer 3: Kreirati Cup specifikaciju za generisanje sintaksnog analizatora jezika µPascal

1. *Program* → **program (** *NizImena* **) ;** *DeklProm Blok* **.**
2. *DeklProm* → **var** *NizDekl*
3. *NizDekl* → *NizDekl Deklaracija*
4. | *Deklaracija*
5. *Deklaracija* → *NizImena* **:** *Tip*
6. *NizImena* → *NizImena* **, ID**
7. | **ID**
8. *Tip* → **integer**
9. | **char**
10. *Blok* → **begin** *NizNar* **end**
11. *NizNar* → *NizNar* **;** *Naredba*
12. | *Naredba*
13. *Naredba* → *Ulaz*
14. | *Izlaz*
15. | *Dodela*
16. | *Blok*
17. | *IfNar*
18. *Ulaz* → **read (ID)**
19. *Izlaz* → **write (** *Izraz* **)**
20. *Dodela* → **ID** := *Izraz*
21. *Izraz* → *Izraz* + *Pizraz*
22. | *Pizraz*
23. *Pizraz* → *Pizraz* * *Fizraz*
24. | *Fizraz*
25. *Fizraz* → **ID**
26. | **CONST**
27. | **(** *Izraz* **)**
28. *IfNar* → **if** *Izraz* **then** *Naredba* **else** *Naredba*

Cup specifikacija