Lavanda

Exercise with Attribute Grammar and its implementation in JavaCC

July 18, 2007

Contents

Chapter 1

Problem

Lavanda is a Domain Specific Language (*DSL*) which main goal is describe the bags of clothes that Point of Gathering of a Laundry daily send to the Center to wash. Each bag has a identification number, the client name and its content is divided in one or more items. Each item have one or more clothe type (personal clothe or *household linen*), tinged type (white or color) and line type (cotton, wool and fiber). For each one of this items we keep in register the number of pieces that belongs to that item. The Independent Context Grammar$G$, mentioned below, defines the language Lavanda intended. The root is `Lavanda`, the terminal simbols are written is lowercase (pseudo-terminais), or uppercase (reserved-words), or between apostrophes (sinais de pontuação) and null string í noted by `&`; the remaining simbols are Non-Terminals.

```
p1:  Lavanda  -->  Cabec  Sacos
p2:  Cabec    -->  data  IdPR
p3:  Sacos    -->  Saco  '.'
p4:              |  Sacos  Saco  '.'
p5:  Saco     -->  num  IdCli  Lotes
p6:  Lotes    -->  Lote  Outros
p7:  Lote     -->  Tipo  Qt
p8:  Tipo     -->  Classe  Tinto  Fio
p9:  Outros   -->  &
p10:             |  ';'  Lotes
p11: IdPR     -->  id
p12: IdCli    -->  id
p13: Qt       -->  num
p14,15:    Classe-->  corpo |  casa
p16,17:    Tinto -->  br    |  cor
p18,19,20: Fio   -->  alg   |  la    |  fib
```

After transform G in a independent contex abstract grammar (you can reduce some productions that seems redundant), writte a **Attribute Grammar** for:

- compute (and print) total of bags sended and total of items of each cliente.
- compute (and print) total of pieces of each 12 items types (since 'body/br/alg' until 'house/cor/fib') sended to wash at laundry.
- compute total cost of each bag; suppose initially is given a table with prices of each item type.

  The grammar should detect error situations: the identification number of bag is duplicated and should flag an error allways show up a bag for a client already finded.

Chapter 2

Attribute Grammar - Solution

The first step is writte the abstract grammar.
To do that we eliminate all terminals without semantic charge (reserved words and signs). The grammar will be simplified by eliminating productions without alternatives that in right side just show up one terminal — in this case: `p11, p12, p13`.

```
p1a:  Lavanda  -->  Cabec  Sacos
p2a:  Cabec    -->  data  id
p3a:  Sacos    -->  Saco
p4a:              |  Sacos  Saco
p5a:  Saco     -->  num  id  Lotes
p6a:  Lotes    -->  Lote  Outros
p7a:  Lote     -->  Tipo  num
p8a:  Tipo     -->  Classe  Tinto  Fio
p9a:  Outros   -->  &
p10a:             |  Lotes
p11a:    Classe-->  corpo
p12a:             |  casa
p13a:    Tinto -->  br
p14a:             |  cor
p15a:    Fio   -->  alg
p16a:             |  la
p17a:             |  fib
```

The next step is choose the attributes.

- For first item, we will need two sinthesized attributes: `nSacos: int` associated at axiom `Lavanda` and `nLotes: int` associated at symbol `Saco`.
  To compute each one will be necessary associate: `nSacos: int` at symbol `Sacos` and `nLotes: int` at symbol `Lotes` and at symbol `Outros`.

  The computation and translate rules are:

```
p1a:  Lavanda  -->  Cabec  Sacos
      -- Lavanda.nSacos = Sacos.nSacos
      -- escreve( Lavanda.nSacos )
p3a:  Sacos    -->  Saco
      -- Sacos.nSacos = 1
p4a:              |  Sacos  Saco
```

```
      -- Sacos0.nSacos = Sacos1.nSacos + 1
p5a:  Saco     -->  num  id  Lotes
      -- Saco.nLotes = Lotes.nLotes
      -- escreve( Saco.nLotes )
p6a:  Lotes    -->  Lote  Outros
      -- Lotes.nLotes = Outros.nLotes + 1
p9a:  Outros   -->  &
      -- Outros.nLotes = 0
p10a:            |  Lotes
      -- Outros.nLotes = Lotes.nLotes
```

- To this item will be needed 3 attributes:

  1. `inEnv: HashTable` — `Saco`, `Lotes` and `Lote`;
  2. `outEnv: HashTable` — `Lavanda`, `Sacos`, `Saco`, `Lotes`, `Lote` and `Outros`;
  3. `name: string` — `Tipo`, `Classe`, `Tinto` and `Fio`.

  The computation and translate rules are:

```
p1a:  Lavanda  -->  Cabec  Sacos
      -- escreveT( Sacos.outEnv )
p3a:  Sacos    -->  Saco
      -- Saco.inEnv = Sacos.inEnv
      -- Sacos.outEnv = Saco.outEnv
p4a:             |  Sacos  Saco
      -- Saco.inEnv = Sacos1.outEnv
      -- Sacos1.inEnv = Sacos0.inEnv
      -- Sacos0.outEnv = Saco.outEnv
p5a:  Saco     -->  num  id  Lotes
      -- Lotes.inEnv = Saco.inEnv
      -- Saco.outEnv = Lotes.outEnv
p6a:  Lotes    -->  Lote  Outros
      -- Lote.inEnv = Lotes.inEnv
      -- Outros.inEnv = Lote.outEnv
      -- Lotes.outEnv = Outros.outEnv
p7a:  Lote     -->  Tipo num
      -- Lote.outEnv = updateTablePrice(Lote.inEnv, Tipo.name, num)
p8a:  Tipo     -->  Classe Tinto Fio
      -- Tipo.name = Classe.name + Tinto.name + Fio.name
p9a:  Outros   -->  &
      -- Outros.outEnv = Outros.inEv;
p10a:            |  Lotes
      -- Lotes.inEnv = Outros.inEnv;
      -- Outros.outEnv = Lotes.outEnv;
p11a: Classe  -->  corpo
      -- Classe.name = "corpo"
p12a: Classe  -->  casa
      -- Classe.name = "casa"
p13a: Tinto  -->  br
      -- Tinto.name = "br"
p14a: Tinto  -->  cor
      -- Tinto.name = "cor"
p15a: Fio  -->  alg
```

```
              -- Fio.name = "alg"
        p16a: Fio  -->  la
              -- Fio.name = "la"
        p17a: Fio  -->  fib
              -- Fio.name = "fib"
```

- To this item will be needed 5 attributes:

  1. `inTable: HashTable` — `Sacos`, `Saco`, `Lotes`, `Lote` and `Outros`;
     Price table (inherited attribute).
  2. `inIds: Vector` — `Sacos` and `Saco`;
     Clients identifiers (Array — inherited attribute).
  3. `outIds: Vector` — `Sacos` and `Saco`;
     Clients identifiers (Array — sinthesized attribute).
  4. `custoTotal: int` — `Saco`, `Lotes`, `Lote` and `Outros`;
     Cost of each bag (sinthesized attribute).
  5. `name: string` —- `Tipo`, `Classe`, `Tinto` and `Fio`. Name of each attribute associated at Tipo (sinthesized attribute).

  The computation and translate rules are:

```
p1a : Lavanda -> Cabec Sacos
              -- Sacos.inTable = initTable()
              -- Sacos.inIds   = initIds()
        p3a:  Sacos    -->  Saco
              -- Saco.inTable = Sacos.inTable
              -- Saco.inIds = Sacos.inIds
              -- Sacos.outIds = Saco.outIds
              -- escrevePreco( Saco.custoTotal )
        p4a:             |  Sacos  Saco
              -- Saco.inTable  = Sacos0.inTable
              -- Sacos1.inEnv = Sacos0.inEnv
              -- Saco.inIds = Sacos1.outIds
              -- Sacos1.inIds = Sacos0.inIds
              -- Sacos0.outIds = Saco.outIds
              -- escrevePreco( Saco.custoTotal )
        p5a:  Saco     -->  num  id  Lotes
              -- Saco.outEnv = novoId( Saco.inIds, num.value() )
              -- if ( pertence( num,Saco.inIds ) )
              --    erro("Cliente ja existente!")
              -- Lotes.inTable   = Saco.inTable
              -- Saco.custoTotal = Lotes.custoTotal
        p6a:  Lotes    -->  Lote  Outros
              -- Lote.inTable = Lotes.inTable
              -- Outros.inTable = Lotes.inTable
              -- Lotes.custoTotal = Lote.custoTotal + Outros.custoTotal
        p7a:  Lote     -->  Tipo num
              -- Lote.custoTotal = lookupPreco( Lote.inEnv, Tipo.name ) * num.value()
        p8a:  Tipo     -->  Classe Tinto Fio
              -- Tipo.name = Classe.name + Tinto.name + Fio.name
        p9a:  Outros   -->  &
              -- Outros.custoTotal = 0
        p10a:            |  Lotes
```

```
      -- Outros.custoTotal = Lotes.custoTotal
p11a: Classe  -->  corpo
      -- Classe.name = "corpo"
p12a: Classe  -->  casa
      -- Classe.name = "casa"
p13a: Tinto  -->  br
      -- Tinto.name = "br"
p14a: Tinto  -->  cor
      -- Tinto.name = "cor"
p15a: Fio  -->  alg
      -- Fio.name = "alg"
p16a: Fio  -->  la
      -- Fio.name = "la"
p17a: Fio  -->  fib
      -- Fio.name = "fib"
```

Chapter 3

JavaCC implementation

"lavanda.jj" 8 ≡

```
options {
  JDK_VERSION = "1.5";
}

PARSER_BEGIN(Lavanda)
package lavanda;

import java.io.*;
import java.util.*;

public class Lavanda {

   static int numSacos = 0;
   static int nLotes = 0;
   static Hashtable<String,Integer> lotesTable = Environment.InitTableMaterials();
   static String name;
   static Hashtable<String, Double> priceTable = Environment.InitTablePrice();
   static Double custoTotal = 0.0;
   static ArrayList<Integer> clientIds = new ArrayList<Integer>();

   public static void main(String[] args){
        try{
          if(args.length > 0){
         Lavanda parser = new Lavanda(new PushbackReader( new FileReader(args[0]), 1024));
               parser.lavanda();
       }
       else{
              System.err.println("Sem argumentos para processar!");
      System.exit(0);
       }
          }
          catch(Exception e){
            System.out.println(e.getMessage());
          }
          catch(TokenMgrError err){
            System.out.println(err.getMessage());
          }
   }
}
PARSER_END(Lavanda)
◇
```

File defined by 8, 9, 10, 11, 12.

"lavanda.jj" 9 ≡

```
SKIP :
{
   " "
|  "\r"
|  "\t"
|  "\n"
}

TOKEN : /* Ajudantes */
{
    <#DIGIT:  ["0" - "9"] >
|   <#DAY:    (["0" - "3"])? <DIGIT> >
|   <#MONTH:  (["0" - "1"])? <DIGIT> >
|   <#YEAR:   ["0" - "2"] (<DIGIT>){3} >
|   <#SEP:    "-" >
|   <#LETTER: ["a" - "z", "A" - "Z"]>
}

TOKEN :
{
     <DATE:   <DAY> <SEP> <MONTH> <SEP> <YEAR> >
|    <NUMBER: ( <DIGIT> )+ >
|    <COMMA:  "," >
|    <R_PAR:  "(" >
|    <L_PAR:  ")" >
|    <DASH:   <SEP> >
|    <COR:    "cor" >
|    <BR:     "br"  >
|    <ALG:    "alg" >
|    <FIB:    "fib" >
|    <LA :    "la"  >
|    <CASA:   "casa" >
|    <CORPO:  "corpo">
|   <ID:     ( <LETTER> )+ >
}

◇
```

File defined by 8, 9, 10, 11, 12.

"lavanda.jj" 10 ≡

```
void lavanda() :
{}
{
    cabec() sacos()
    {
      //Prints
      System.out.println(Environment.printTable(lotesTable));
      System.out.println("Total de Sacos: " + numSacos);
    }
}

void cabec() :
{}
{
    <DATE> <ID>
}

void sacos() :
{}
{
   (
      saco()
      {
         //Item 1
         numSacos += 1;

         //Item 3
         System.out.println("Custo Total do Saco: " + custoTotal);
         custoTotal = 0.0;
      }
   )+

}
```

◇

File defined by 8, 9, 10, 11, 12.

"lavanda.jj" 11 ≡

```
void saco() :
{
   Token idSaco;
   nLotes = 0;
}
{
   idSaco = <NUMBER> <ID> "(" lotes() ")"
   {

      //Item 3
      Integer id = new Integer(idSaco.toString()).intValue();
      if(clientIds.contains(id)){
         System.err.println("Cliente " + id + " est repetido!");
         System.exit(0);
      }
      clientIds.add(id);

      //Item 1 - Prints
      System.out.println("\nSaco n. " + idSaco.toString() + " tem " + nLotes + " lotes!");

   }
}

void lotes() :
{}
{
   lote() outros()
   {
      //Item 1
      nLotes += 1;
   }
}

void lote() :
{
   Token t;
}
{
   name = tipo() t = <NUMBER>
   {
      //Item2
      Integer ntab = lotesTable.get(name);
      Integer n = new Integer(t.toString()).intValue();
      lotesTable.put(name, n + ntab);

      //Item3
      custoTotal += priceTable.get(name) * n;
   }
}
◇
```

File defined by 8, 9, 10, 11, 12.

"lavanda.jj" 12 ≡

```
void outros() :
{}
{
   ( "," lotes() )?

}

String tipo() :
{
   String c, t, f;
}
{
   c = classe() <DASH>
   t = tinto() <DASH>
   f = fio()
   {
      //Item 2
      return (c + "-" + t + "-" + f);
   }
}

String classe() :
{}
{
   <CORPO> {return "corpo";}
|  <CASA>  {return "casa"; }
}

String tinto() :
{}
{
   <BR>  {return "br";}
|  <COR> {return "cor";}
}

String fio() :
{}
{
   <ALG> {return "alg";}
|  <FIB> {return "fib";}
|  <LA>  {return "la";}
}
```

◇

File defined by 8, 9, 10, 11, 12.

Chapter 4

Example test

"test.txt" 13 ≡

```
17-07-2007
Lavanda
1 Dani   (corpo-cor-la 1, casa-cor-alg 2)
2 Celina (casa-br-fib 4)
3 Pedro  (corpo-cor-alg 2, corpo-cor-la 3, corpo-cor-fib 1,
          casa-cor-alg 2, casa-cor-la 3, casa-cor-fib 1)

◇
```

Chapter 5

Results

```
C:\java -jar lavanda.jar teste.txt

Saco no. 1 tem --> 2 Lotes!
Custo total do saco: 9.5

Saco no. 2 tem --> 1 Lotes!
Custo total do saco: 28.4

Saco no. 3 tem --> 6 Lotes!
Custo total do saco: 40.6

casa-br-la      0
corpo-br-fib    0
casa-br-alg     0
corpo-cor-la    4
corpo-cor-alg   2
casa-cor-alg    4
casa-br-fib     4
corpo-br-la     0
casa-cor-la     3
corpo-br-alg    0
corpo-cor-fib   1
casa-cor-fib    1

Total de Sacos: 3
```

Chapter 6

Debug

One of the features that JavaCC offers to the user, is its debugger System. One can debug the Parser, the Token Manager or even the Lookhaead.

The next example shows how the Parser's debugger works.

As input to the compiler, we pass a file with the following text:

```
17-7-2007
Lavanda
1 Nuno (corpo-cor-fib 7)
```

The Parser's debugger will create an hierarchy, which root symbol is the root of the Parser (Nonterminal lavanda). Then it calls the first symbol on the rihgt-hand side of the first production. In our case it is the nonterminal cabec. Now it will be parsing the symbols of the cabec's production, and since they are just terminal symbols, it will consume them, or output a parse error whether the lexer matches the words in the text file or not.

After parse those terminals, it calls the second symbol on the first production. Since it is a nonterminal, the debugger starts parsing the new nonterminal's production, calling its symbols, and doing the same work it did for the first production.

At the end, if the text matches the grammar structure, the debugger will stop, returning the root of the parser.

```
Call:   lavanda
  Call:   cabec
    Consumed token: <<DATE>: "17-7-2007" at line 1 column 1>
    Consumed token: <<ID>: "Lavanda" at line 2 column 1>
  Return: cabec
  Call:   sacos
    Call:   saco
      Consumed token: <<NUMBER>: "1" at line 3 column 1>
      Consumed token: <<ID>: "Nuno" at line 3 column 3>
      Consumed token: <"(" at line 3 column 8>
      Call:   lotes
        Call:   lote
          Call:   tipo
```

```
            Call:   classe
              Consumed token: <"corpo" at line 3 column 9>
            Return: classe
            Consumed token: <<DASH>: "-" at line 3 column 14>
            Call:   tinto
              Consumed token: <"cor" at line 3 column 15>
            Return: tinto
            Consumed token: <<DASH>: "-" at line 3 column 18>
            Call:   fio
              Consumed token: <"fib" at line 3 column 19>
            Return: fio
          Return: tipo
          Consumed token: <<NUMBER>: "7" at line 3 column 23>
        Return: lote
        Call:   outros
        Return: outros
      Return: lotes
      Consumed token: <")" at line 3 column 24>
    Return: saco
  Return: sacos
Return: lavanda
```

Chapter 7

Files