Lavanda

Exercise with Attribute Grammar and its implementation in Lex/Yacc

May 2, 2006

Contents

Chapter 1

Problem

Lavanda is a Domain Specific Language (*DSL*) which main goal is drescribe 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), ou 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 grammarr 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

Lex/Yacc implementation

3.1 Lex

"Lavanda.l" 7 ≡

```
%{
#include <stdio.h>
#include <string.h>
#include "y.tab.h"
%}

digit   [0-9]
day     [0-3]?{digit}
month   [0-1]?{digit}
year    [0-2]{digit}{digit}{digit}
sep     [//|-]
data    {day}{sep}{month}{sep}{year}

%%

{digit}*                { yylval.number=atoi(yytext); return NUMBER; }
{data}                  { yylval.string=strdup(yytext); return DATA; }
"alg"                   { yylval.string=strdup(yytext); return ALG; }
"br"                    { yylval.string=strdup(yytext); return BR;   }
"cor"                   { yylval.string=strdup(yytext); return COR; }
"casa"                  { yylval.string=strdup(yytext); return CASA; }
"corpo"                 { yylval.string=strdup(yytext); return CORPO; }
"fib"                   { yylval.string=strdup(yytext); return FIB; }
"la"                    { yylval.string=strdup(yytext); return LA; }
[a-z]+                  { yylval.string=strdup(yytext); return IDENT; }
[ \n\t\f]               { /* white space is skipped */ }
[-.;,()?]               { return yytext[0]; }
.         ;

%%

int yywrap() { return 1; }

◇
```

3.2 Yacc

"Lavanda.y" 8 ≡

```
%{
#include <string.h>
#include <stdio.h>
#include "hashtable.c"
#include "list.c"

HashTable nLotes[TABSIZE], tablePrice[TABSIZE];
List idSacos;
float custoTotal = 0;

%}


%union
{
        int number;
   char *string;
}

%token <number> NUMBER
%token <string> IDENT DATA CORPO CASA COR BR LA FIB ALG

%type <number> Sacos Saco Lotes Outros
%type <string> Tipo Classe Tinto Fio

%start Lavanda

%%


Lavanda:
       Cabec Sacos         { printf("\nTabela Lotes:\n");
                 printHashTable(nLotes);
                 printf("\nTotal Sacos: %d\n", $2);
                 exit(0);
}
       ;

Cabec:
     DATA IDENT
     ;

Sacos:
     Saco '.'           { $$ = 1;         }
   | Sacos Saco '.'        { $$ = $1 + 1;       }
   ;

◇
```

File defined by 8, 9, 10.

"Lavanda.y" 9 ≡

```
Saco:
    NUMBER IDENT '(' Lotes ')'      { if (searchList(idSacos,$1)==0) {
                 printf("Saco repetido!\n");
                 exit(0);
                }
                idSacos = insertList(idSacos,$1);
                printf("\nSaco n. %d tem --> %d lotes!\n", $1, $4);
                printf("Custo total do saco: %.2f.\n", custoTotal);
                custoTotal = 0;    }
    ;

Lotes:
     Lote Outros        { $$ = $2 + 1;       }
     ;

Lote:
    Tipo NUMBER            { updateTableNLotes(nLotes,$1,$2);
                custoTotal += searchHashTable(tablePrice,$1)*$2;
             }
    ;


Outros:              { $$ = 0;         }
      | ';' Lotes       { $$ = $2;        }
      ;


Tipo:

    Classe '-' Tinto '-' Fio  { char *aux = strcat($1,"-");
              aux = strcat(aux, $3);
              aux = strcat(aux,"-");
              $$ = strcat(aux,$5);        }
    ;

Classe:
      CORPO       { $$ = $1;     }
    | CASA        { $$ = $1;     }
    ;

Tinto:
     BR           { $$ = $1;     }
    | COR         { $$ = $1;     }
    ;

Fio:
   ALG            { $$ = $1;     }
 | FIB            { $$ = $1;     }
 | LA          { $$ = $1;     }
 ;



%%

◇
```

File defined by 8, 9, 10.

"Lavanda.y" 10 ≡

```
void initNLotes(HashTable env[])
{
   newHashTable(env);
   insertHashTable(env,"corpo-br-la",0);
         insertHashTable(env,"corpo-br-alg",0);
         insertHashTable(env,"corpo-br-fib",0);
         insertHashTable(env,"corpo-cor-la",0);
         insertHashTable(env,"corpo-cor-alg",0);
         insertHashTable(env,"corpo-cor-fib",0);
   insertHashTable(env,"casa-br-la",0);
         insertHashTable(env,"casa-br-alg",0);
         insertHashTable(env,"casa-br-fib",0);
         insertHashTable(env,"casa-cor-la",0);
         insertHashTable(env,"casa-cor-alg",0);
         insertHashTable(env,"casa-cor-fib",0);
}


void initTablePrice(HashTable env[])
{
   newHashTable(env);
   insertHashTable(env,"corpo-br-la",1.0);
   insertHashTable(env,"corpo-br-alg",2.2);
   insertHashTable(env,"corpo-br-fib",3.4);
   insertHashTable(env,"corpo-cor-la",4.5);
   insertHashTable(env,"corpo-cor-alg",3.7);
   insertHashTable(env,"corpo-cor-fib",1.9);
   insertHashTable(env,"casa-br-la",2.6);
   insertHashTable(env,"casa-br-alg",5.3);
   insertHashTable(env,"casa-br-fib",7.1);
   insertHashTable(env,"casa-cor-la",3.5);
   insertHashTable(env,"casa-cor-alg",2.5);
   insertHashTable(env,"casa-cor-fib",2.3);

}


int yyerror(char* error) {
   fprintf(stdout,"Error: %s\n", error);
   return 0;
}

int main() {

  initNLotes(nLotes);
  initTablePrice(tablePrice);
  idSacos = createList();
  return yyparse();

}
```

◇

File defined by 8, 9, 10.

Chapter 4

Example test

"Test.txt" 11 ≡

```
10-11-2005 today  1 dani   (corpo-cor-la 1 , casa-cor-alg 2)
                  2 pedro  (casa-br-fib 4)
                  3 celina (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

```
[_localhost_Lavanda]# ./parser < test1.test

Saco n. 1 tem --> 2 lotes!
Custo total do saco: 9.50.

Saco n. 2 tem --> 1 lotes!
Custo total do saco: 28.40.

Saco n. 3 tem --> 6 lotes!
Custo total do saco: 40.60.

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

Total Sacos: 3
```

Chapter 6

Files