Processamento de Linguagens I
LESI + LMCC (3° ano)

Exercícios com Gramáticas de Atributos
Sua implementação em CoCo/R, LISA, Lex/Yacc, LRC e AntLR

Language.Processing@di.um.pt
23 de Maio de 2006

Conteúdo

Prefácio

Este documento, começou a ser escrito em Agosto de 2005, e é da autoria de Daniela Carneiro da Cruz e Pedro Rangel Henriques, responsáveis pela sua integração e manutenção, tendo contudo o contributo de outros membros do gEPL[1], nomeadamente do Rodrigo Baptista.

O seu objectivo é duplo:

- por um lado, pretendemos mostrar a resolução de um problema de definição de uma linguagem e desenvolvimento do respectivo processador usando Gramáticas de Atributos (GA), mas de uma forma teórica e genérica; para tal discutir-se-á a escolha dos atributos necessários e a escrita das regras de cálculo, das condições de contexto e das regras de tradução.
- por outro lado, pretendemos ilustrar a sua implementação com diferentes ferramentas de geração de compiladores baseadas em GAs.

Para o efeito o documento está organizado em vários capítulos.
No primeiro encontramos o enunciado de um problema que se pretende que seja, por um lado curto e claro, mas por outro que seja um caso real/útil; a sua resolução deve ser também clara e fácil de assimilar para todos aqueles que se estejam a inicar na área.
Após a descrição do problema, passamos à sua resolução abstracta.
Posteriormente, nos capítulos seguintes, apresentamos a implementação dessa GA genérica em diferentes geradores de compiladores.
Por fim são apresentados os resultados obtidos para um mesmo texto-fonte exemplo com cada um dos geradores de compiladores testados nos capítulos precendentes.

[1] O grupo de Especificação e Processamento de Linguagens do Departamento de Informática da Universidade do Minho, www.di.uminho.pt/ gepl.

Capítulo 1

Enunciado do exercício

Lavanda é uma Linguagem de Domínio Específico (DSL) que se destina a descrever as remessas de sacos de roupa que os Pontos de Recolha (PR) de uma Lavandaria enviam diariamente á Central (LC) para lavar. Cada saco tem um número de identificação e o nome do cliente que o deixou; o seu conteúdo está dividido em lotes. Cada lote corresponde a um tipo de peça (roupa-de-corpo, ou roupa-de-casa), um tipo de tinto (branco, ou cor) e um tipo de fio (algodão, lã e fibra), registando-se então o número de peças entregues que pertencem a esse lote.
A gramática independente de contexto G, abaixo apresentada, define a linguagem Lavanda pretendida. O Símbolo Inicial é `Lavanda`, os Símbolos Terminais são escritos em minúsculas (pseudo-terminais), ou em maiúscula (palavras-reservadas), ou entre apostrofes (sinais de pontuação) e a string nula é denotada por `&`; os restantes serão os Símbolos Não-Terminais.

```
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
```

Depois de transformar G numa gramática independente de contexto abstracta *Gabs* (pode reduzir algumas produções que lhe pareçam supérfluas), escreva uma **Gramática de Atributos**, GA, para:

1. calcular (e imprimir) o número total de sacos enviados e número de lotes de cada cliente.
2. calcular e imprimir o total de peças de cada um dos 12 tipos de lotes (desde 'corpo/br/alg' até 'casa/cor/fib') enviadas, para lavar, pelo PR à LC.
3. calcular o custo total de cada saco, supondo que inicialmente é fornecida a tabela de preços do PR em causa (a tabela indica o preço por peça para cada tipo de lote).
   Nesta fase, deve haver ainda a preocupação de detectar situações de erro em que o número

de identificação do saco seja repetido ou sempre que apareça um saco para um cliente já encontrado.

Capítulo 2

Resolução do exercício

Nesta primeira fase, escrevemos a gramática abstracta *Gabs*.
Para tal eliminam-se todos os terminais sem carga semântica (palavras-reservadas e sinais). A *G* será simplificada eliminando produções sem alternativas em que no lado direito só aparece um terminal — neste caso: `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
```

Para cada uma das três alíneas , a resoluçã é feita em 2 passos.

O primeiro passo é escolher os atributos (classe, tipo) a associar aos símbolos para extrair a informação pedida.
O segundo passo é escrever as regras de cálculo para os símbolos respectivos.

2.1 Alínea 1

Para esta alínea serão precisos 2 atributos sintetizados:

1. `nSacos::int` associados a `Lavanda` e `Sacos`;
2. `nLotes::int` associados a `Saco`, `Lotes` e `Outros`.

As regras de cálculo e as regras de tradução a associar às produçoes onde os símbolos em causa aparecem são:

```
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
```

2.2 Alínea 2

Para esta alínea serão precisos 3 atributos:

1. `inEnv::HashTable` a `Saco`, `Lotes` e `Lote` — atributo herdado;
2. `outEnv::HashTable` a `Lavanda`, `Sacos`, `Saco`, `Lotes`, `Lote` e `Outros` — atributo sintetizado;
3. `name::String` a `Tipo`, `Classe`, `Tinto` e `Fio` — atributo sintetizado.

As regras de cálculo e as regras de tradução a associar às produções onde os símbolos em causa aparecem são:

```
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"
```

2.3 Alínea 3

Para esta alínea serão precisos 5 atributos:

1. `inTable::HashTable` aos símbolos `Sacos`, `Saco`, `Lotes`, `Lote` e `Outros` — atributo herdado (Tabela de prec cos);
2. `inIds::Vector` aos símbolos `Sacos` e `Saco` — atributo herdado (Array de identificadores de clientes);
3. `outIds::Vector` aos símbolos `Sacos` e `Saco` — atributo sintetizado (Array de identificadores de clientes);
4. `custoTotal::int` aos símbolos `Saco`, `Lotes`, `Lote` e `Outros` — atributo sintetizado (Custo de cada saco);
5. `name::string` aos símbolos `Tipo`, `Classe`, `Tinto` e `Fio` — atributo sintetizado (Nome de cada atributo associado ao símbolo Tipo).

As regras de cálculo e as regras de tradução a associar às produções onde os símbolos em causa aparecem são:

```
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  --> 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("Numero de saco repetido!")
      -- 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"
```

Capítulo 3

Implementação CoCoR

3.1 Gramática - Scanner, Parser & Evaluator

Toda a gramática de atributos, incluindo a especificação léxica, é escrita num único ficheiro ("Lavanda.atg"), que se lista a seguir. Os atributos são declarados em cada produção junto dos respectivos sÃmbolos entre '<' e '>'. As regras de c'alculo s ao escritas em C# ao longo da produção entre "(."e ".)".

"Lavanda.atg" 9 ≡

```
COMPILER Lavanda

/*************************************************************************
*        SCANNER                    *
*************************************************************************/

COMPILER Lavanda

public OpHashtable opTable;

// ---------------

IGNORECASE

CHARACTERS
letter = 'a'..'z'.
digit  = '0'..'9'.

TOKENS
ident = letter { letter }.
number = digit { digit }.


IGNORE '\r' + '\n' + '\t'

◇
```

File defined by 9, 10, 11.

"Lavanda.atg" 10 ≡

```
/*************************************************************
*        PARSER                          *
*************************************************************/

PRODUCTIONS

/*--------------------------- PRODUCTION FOR GRAMMAR NAME -----------------*/

Lavanda              (. int nSacos = 0;
                     Hashtable inTable = opTable.initNLotes(),
                outTable,
                inTPrice = opTable.initTablePrice();
                ArrayList inIds = opTable.initIds(),
                outIds = inIds;         .)
= Cabec Sacos<out nSacos, inTable, out outTable, inTPrice, inIds, out outIds>
            (. Console.WriteLine("Total sacos: " +
                        nSacos + "\n");
                opTable.printTableLotes(outTable);
                           .).

Cabec
= Data ident.


Data           (. int day, month, year;      .)
= number          (. day = Convert.ToInt32(t.val);
                if (day < 1 || day > 31)
                SemErr("Wrong day!");      .)
  '-'
  number          (. month = Convert.ToInt32(t.val);
                if (month < 1 || month > 12)
                SemErr("Wrong month!");    .)
  '-' number         (. year = Convert.ToInt32(t.val);
                   if (year < 2000 || day > 2010)
                SemErr("Wrong year!");     .).



Sacos<out int nSacos, Hashtable inTable,out Hashtable outTable,
      Hashtable inTPrice,
      ArrayList inIds, out ArrayList outIds>
            (. int nSacos1 = 0;
               float custoTotal;    .)
=   Saco<inTable,out outTable, inTPrice, out custoTotal, inIds, out outIds>
            (. nSacos = 1;       .)
   [ Sacos<out nSacos1, outTable, out outTable, inTPrice, outIds, out outIds>
            (. nSacos += nSacos1;         .) ].



Saco<Hashtable inTable,out Hashtable outTable,Hashtable inTPrice,
     out float custoTotal, ArrayList inIds, out ArrayList outIds>
            (. int nLotes = 0, num;
               custoTotal = 0;         .)

= number          (. num = Convert.ToInt32(t.val); .)
  ident '(' Lotes<out nLotes, inTable,out outTable, inTPrice, out custoTotal>
   ')'
            (. Console.WriteLine("N. lotes no saco " +
                                                num + ": " + nLotes);
               opTable.writePrice(custoTotal);
               if ( inIds.Contains( num ))
               throw new Exception("Number of Bag already exists!");
               outIds = opTable.newId(inIds, num);
```

"Lavanda.atg" 11 ≡

```
Lotes<out int nLotes, Hashtable inTable,out Hashtable outTable,
      Hashtable inTPrice, out float custoTotal>
            (. int nLotes1 = 0;
               float custoTotal2 = 0;     .)
= Lote<inTable,out outTable, inTPrice, out custoTotal>
  Outros<out nLotes1, outTable, out outTable, inTPrice, out custoTotal2>
            (. nLotes = nLotes1 +   1;
               custoTotal += custoTotal2;    .).


Lote<Hashtable inTable,out Hashtable outTable,
     Hashtable inTPrice, out float custoTotal>
            (. string name = "";       .)
= Tipo<out name> number
            (. outTable = opTable.updateTableLotes(
                     inTable, name, Convert.ToInt32(t.val));
               custoTotal = opTable.lookupPrice(inTPrice, name) *
                                              Convert.ToInt32(t.val);
                          .).


Outros<out int nLotes, Hashtable inTable,out Hashtable outTable,
       Hashtable inTPrice, out float custoTotal>
                (. nLotes = 0;
                   outTable = inTable;
              custoTotal = 0;       .)
= [ ',' Lotes<out nLotes, inTable,out outTable, inTPrice, out custoTotal> ].


Tipo<out string name>      (. string name1 = "",
               name2 = "",
               name3 = "";          .)
= Classe<out name1> '-'
  Tinto<out name2> '-'
  Fio<out name3>     (. name = name1 + "-" +
                                       name2 + "-" +
                                       name3;         .).


Classe<out string name>
         (. name = "";     .)
=   "corpo"    (. name = "corpo";   .)
  | "casa"     (. name = "casa"; .).

Tinto<out string name>
         (. name = "";     .)
= "br"         (. name = "br";      .)
 | "cor"               (. name = "cor";  .).

Fio<out string name>
         (. name = "";     .)
= "alg"        (. name = "alg";  .)
 | "la"                (. name = "la";      .)
 | "fib"               (. name = "fib";  .).


END Lavanda.
◇
```

File defined by 9, 10, 11.

3.2 Funções auxiliares

As funções auxiliares sobre tabelas de Hash (nomeadamente tabela de preços foram definidas no seguinte módulo:

"OpHashtable.cs" 12 ≡

```
using System;
using System.Collections;

namespace Lavanda
{
   public class OpHashtable
   {
     public OpHashtable()
        {   }

     // alinea b
     public Hashtable initNLotes()
     {
         Hashtable env = new Hashtable();
         env.Add("corpo-br-la",0);
         env.Add("corpo-br-alg",0);
         env.Add("corpo-br-fib",0);
         env.Add("corpo-cor-la",0);
         env.Add("corpo-cor-alg",0);
         env.Add("corpo-cor-fib",0);
         env.Add("casa-br-la",0);
         env.Add("casa-br-alg",0);
         env.Add("casa-br-fib",0);
         env.Add("casa-cor-la",0);
         env.Add("casa-cor-alg",0);
         env.Add("casa-cor-fib",0);

         return env;
     }

   public Hashtable updateTableLotes(Hashtable inTable, String name, int number)
   {
     IDictionaryEnumerator env = inTable.GetEnumerator();

     while(env.MoveNext())
     {
      string key = (string)env.Key;
      if (key.Equals(name))
         {
         int pieces = (int)env.Value + number;
         inTable.Remove(key);
         inTable.Add(key, pieces);
         break;
         }
      }
    return inTable;
    }

}
```

◇

File defined by 12, 13, 14.

"OpHashtable.cs" 13 ≡

```
 public void printTableLotes( Hashtable myList )
 {
  IDictionaryEnumerator myEnumerator = myList.GetEnumerator();
  Console.WriteLine( "\t-Descricao-\t-N. Lotes-" );
  while ( myEnumerator.MoveNext() )
   Console.WriteLine("\t{0}:\t{1}", myEnumerator.Key, myEnumerator.Value);
   Console.WriteLine();
  }


      // alinea c

public Hashtable initTablePrice()
{
Hashtable env = new Hashtable();
env.Add("corpo-br-la",1.0f);
env.Add("corpo-br-alg",2.2f);
env.Add("corpo-br-fib",3.4f);
env.Add("corpo-cor-la",4.5f);
env.Add("corpo-cor-alg",3.7f);
env.Add("corpo-cor-fib",1.9f);
env.Add("casa-br-la",2.6f);
env.Add("casa-br-alg",5.3f);
env.Add("casa-br-fib",7.1f);
env.Add("casa-cor-la",3.5f);
env.Add("casa-cor-alg",2.5f);
env.Add("casa-cor-fib",2.3f);

return env;
}



public float lookupPrice (Hashtable inPrice, string name )
{
float price = 0;
IDictionaryEnumerator env = inPrice.GetEnumerator();
while ( env.MoveNext() )
{
if (env.Key.Equals(name))
   {
   price = (float)env.Value;
   break;
   }
}
return price;
}
```

◇

File defined by 12, 13, 14.

"OpHashtable.cs" 14 ≡

```
        public ArrayList initIds()
        {
        return ( new ArrayList() );
        }


        public ArrayList newId(ArrayList old, int num)
        {
        old.Add(num);
        return ( (ArrayList)old.Clone() );
        }

        public void writePrice(float num)
        {
        Console.WriteLine("Preco Total: " + num + "\n" );
        }
       }
      } // end namespace
```

◇

File defined by 12, 13, 14.

Capítulo 4

Implementação LISA

No caso do gerador LISA, descreve-se toda a gramática de atributos, incluindo a especificação léxica num ficheiro único — Lavanda.lisa — apresentado a seguir.
Os atributos são declarados explicitamente uma vez, antes das produções.
As regras de cálculo escrevem-se em Java no fim de cada produção num bloco "{ }"próprio.
O código dos métodos Java auxiliares são incluídos no mesmo ficheiro, no fim das regras semânticas.

"Lavanda.lisa" 15 ≡

```
language Lavanda
{
   lexicon
    {
        ReservedWord corpo | casa | br | cor | alg | fib | la
        Number     [0-9]+
        Data        [0-2][0-9]\-[0-9][0-9]\-[0-2][0-9][0-9][0-9]
        Identifier [a-z]+
        separa \( | \) | \, | \> | \-
        ignore      [\0x09\0x0A\0x0D\  ]+
    }


attributes int       LAVANDA.nSacos, CABEC.nSacos, SACOS.nSacos,
                     SACO.nLotes, LOTES.nLotes, LOTE.nLotes, OUTROS.nLotes;
           String    TIPO.name, CLASSE.name, TINTO.name, FIO.name,
                     LAVANDA.output, SACOS.output, SACO.output;
           // b)
           Hashtable SACOS.inTable, SACO.inTable, OUTROS.inTable,
                     LOTE.inTable, LOTES.inTable,
           SACOS.outTable, SACO.outTable, OUTROS.outTable,
                     LOTE.outTable, LOTES.outTable;
    // c)
    Hashtable  SACOS.inTablePrice, SACO.inTablePrice, LOTES.inTablePrice,
                     LOTE.inTablePrice, OUTROS.inTablePrice;
    Vector     SACOS.inIds, SACO.inIds,
                     SACOS.outIds, SACO.outIds;
    double     SACO.custoTotal, LOTES.custoTotal, OUTROS.custoTotal, LOTE.custoTotal;

◇
```

File defined by 15, 16, 17, 18, 19, 20, 21.

"Lavanda.lisa" 16 ≡

```
      rule Lavanda {
       LAVANDA ::= CABEC SACOS compute {
          LAVANDA.nSacos = SACOS.nSacos;
          LAVANDA.output = "\n\nSACOS:" + escreve( LAVANDA.nSacos, "sacos" ) +
                           "\n\nLOTES: " + SACOS.output + "\n" +
                          "TABELA LOTES: " + printTable(SACOS.outTable);

          // b)
          SACOS.inTable = initNLotes();

          // c)
          SACOS.inTablePrice = initTablePrice();
          SACOS.inIds        = initIds();
                      };
       }

    rule Cabec {
           CABEC ::= #Data #Identifier compute {
            CABEC.nSacos = 0;
            };
               }


       rule Sacos1 {
            SACOS ::= SACO compute {
            SACOS.nSacos = 1;
            SACOS.output = SACO.output + writePrice( SACO.custoTotal );

            // b)
            SACO.inTable   = SACOS.inTable;
            SACOS.outTable = SACO.outTable;

            // c)
            SACO.inTablePrice = SACOS.inTablePrice;
            SACO.inIds        = SACOS.inIds;
            SACOS.outIds      = SACO.outIds;
            };
               }
```

◇

File defined by 15, 16, 17, 18, 19, 20, 21.

"Lavanda.lisa" 17 ≡

```
rule Sacos2 {
      SACOS ::= SACOS SACO compute {
       SACOS[0].nSacos = SACOS[1].nSacos + 1;
       SACOS[0].output = SACOS[1].output + SACO.output + writePrice( SACO.custoTotal );

       // b)

       SACO.inTable      = SACOS[1].outTable;
       SACOS[1].inTable  = SACOS[0].inTable;
       SACOS[0].outTable = SACO.outTable;

       // c)
       SACO.inTablePrice      =  SACOS[0].inTablePrice;
       SACOS[1].inTablePrice  =  SACOS[0].inTablePrice;
       SACO.inIds             =  SACOS[1].outIds;
       SACOS[1].inIds         =  SACOS[0].inIds;
       SACOS[0].outIds        =  SACO.outIds;

       };
}

 rule Saco {
      SACO ::= #Number #Identifier \( LOTES \) compute {
       SACO.nLotes = LOTES.nLotes;
       SACO.output = escreve( SACO.nLotes, "lotes" );

       // b)
       LOTES.inTable  = SACO.inTable;
       SACO.outTable  =  LOTES.outTable;

       // c)
       SACO.outIds         = newId( SACO.inIds, Integer.valueOf( #Number.value()).intValue() );
       LOTES.inTablePrice  = SACO.inTablePrice;
       SACO.custoTotal     = LOTES.custoTotal;
       if ( inIds.contains( Integer.valueOf(#Number.value()).intValue() ))
       throw new Exception("Cliente already exists!");
       };
          }


rule Lotes {
 LOTES ::= LOTE OUTROS compute {
    LOTES.nLotes = OUTROS.nLotes + 1;

    // b)
    LOTE.inTable    =  LOTES.inTable;
    OUTROS.inTable  =  LOTE.outTable;
    LOTES.outTable  =  OUTROS.outTable;

    // c)

    LOTE.inTablePrice    =  LOTES.inTablePrice;
    OUTROS.inTablePrice  =  LOTES.inTablePrice;
    LOTES.custoTotal     =  LOTE.custoTotal + OUTROS.custoTotal;
```

◇

File defined by 15, 16, 17, 18, 19, 20, 21.

"Lavanda.lisa" 18 ≡

```
           };
        }

       rule Lote {
            LOTE ::= TIPO #Number compute {
                     LOTE.nLotes = 0;

                     // b)
                     LOTE.outTable = updateTablePrice(LOTE.inTable, TIPO.name,
                                     Integer.valueOf(#Number.value()).intValue());

                     // c)
                     LOTE.custoTotal = lookupPrice( LOTE.inTablePrice, TIPO.name ) *
                                       (Integer.valueOf(#Number.value()).intValue());

                     };
                   }

       rule Tipo {
            TIPO ::= CLASSE \- TINTO \- FIO compute {
                     TIPO.name = CLASSE.name + "/" + TINTO.name + "/" + FIO.name;
                     };
                    }


      rule Outros {
            OUTROS ::= compute {
                     OUTROS.nLotes = 0;

                     // b)
                     OUTROS.outTable = OUTROS.inTable;

                     // c)
                     OUTROS.custoTotal = 0;

                     }
                   | \, LOTES compute {
                     OUTROS.nLotes = LOTES.nLotes;

                     // b)
                     LOTES.inTable = OUTROS.inTable;
                     OUTROS.outTable = LOTES.outTable;

                     // c)
                     OUTROS.custoTotal = LOTES.custoTotal;
                     LOTES.inTablePrice = OUTROS.inTablePrice;
                     };
                    }
```

◇

File defined by 15, 16, 17, 18, 19, 20, 21.

"Lavanda.lisa" 19 ≡

```
rule Classe {
    CLASSE ::= corpo compute {
             CLASSE.name = "corpo";
             }
            |  casa compute {
               CLASSE.name = "casa";
            };
            }


rule Tinto {
    TINTO ::= br compute {
             TINTO.name = "br";
             }
            | cor compute {
             TINTO.name = "cor";
             };
            }

rule Fio {
    FIO ::= alg compute {
             FIO.name = "alg";
             }
           | la compute {
             FIO.name = "la";
             }
           | fib compute {
             FIO.name = "fib";
             };
            }


method Print
 {
   import java.util.*;


   public String escreve(int num, String descripton)
   {
      String str = "\n\nNumero de " + descripton + ": " + num;
      return str;
   }


   // b)
```

◇

File defined by 15, 16, 17, 18, 19, 20, 21.

"Lavanda.lisa" 20 ≡

```
public Hashtable initNLotes()
 {
   Hashtable env = new Hashtable();
        env.put("corpo/br/la",0);
              env.put("corpo/br/alg",0);
              env.put("corpo/br/fib",0);
              env.put("corpo/cor/la",0);
              env.put("corpo/cor/alg",0);
              env.put("corpo/cor/fib",0);
        env.put("casa/br/la",0);
              env.put("casa/br/alg",0);
              env.put("casa/br/fib",0);
              env.put("casa/cor/la",0);
              env.put("casa/cor/alg",0);
              env.put("casa/cor/fib",0);

              return env;
  }


  public Hashtable updateTablePrice(Hashtable inTable, String name, int number)
 {
        inTable = (Hashtable)inTable.clone();
        int pieces = ((Integer)inTable.get(name)).intValue();
   inTable.remove(name);
   inTable.put(name,number+pieces);
   return inTable;
 }

  public String printTable(Hashtable inTable)
  {
    String out="\n\n", str="";

    for (Enumeration et = inTable.keys(); et.hasMoreElements();)
    {
      str = (String)et.nextElement();
      int pieces = ((Integer)inTable.get(str)).intValue();
      out +=  str + " ----> " + pieces + "\n";
    }

    return out;
  }
```

◇

File defined by 15, 16, 17, 18, 19, 20, 21.

"Lavanda.lisa" 21 ≡

```
              // c)

              public Hashtable initTablePrice()
             {
               Hashtable env = new Hashtable();
                     env.put("corpo/br/la",1.0);
                           env.put("corpo/br/alg",2.2);
                           env.put("corpo/br/fib",3.4);
                           env.put("corpo/cor/la",4.5);
                           env.put("corpo/cor/alg",3.7);
                           env.put("corpo/cor/fib",1.9);
                     env.put("casa/br/la",2.6);
                           env.put("casa/br/alg",5.3);
                           env.put("casa/br/fib",7.1);
                           env.put("casa/cor/la",3.5);
                           env.put("casa/cor/alg",2.5);
                           env.put("casa/cor/fib",2.3);

                           return env;
               }

            public double lookupPrice ( Hashtable in, String name )
            {
                    return ( ((Double)in.get(name)).doubleValue() );
            }

                      public Vector initIds()
                      {
                        return ( new Vector() );
                      }

            public Vector newId(Vector old, int num)
                      {
                        old.addElement(num);
                        return ( (Vector)old.clone() );
                      }

            public String writePrice(double num)
            {
               return ( "\n\nPreco Total: " + num + "\n" );
            }


         }
      ◇
```

File defined by 15, 16, 17, 18, 19, 20, 21.

Capítulo 5

Implementação Lex/Yacc

Neste caso, não se usa propriamente uma gramática de atributos, mas sim uma gramática tradutora que apenas tem atributos sintetizados (um por cada símbolo).

5.1 Lex

A descrição léxica é feita à frente da `GramÃ<tica Tradutora` no ficheiro "Lavanda.l"que será tratada pela ferramenta Flex.

"Lavanda.l" 22 ≡

```
%{
#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}

%%
```

◇

File defined by 22, 23a.

"Lavanda.l" 23a ≡

```
{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; }
```

◇

File defined by 22, 23a.

5.2 Yacc

Quanto à gramática tradutora, é escrita num único ficheiro — Lavanda.y — para ser processado pela ferramenta Yacc. O tipo do atributo sintetizado que se pode associar a cada símbolo e a respectiva associação será feita no início antes da gramática.
As acções semânticas, agora especificadas em C, são escritas entre '{' e '}' ao longo de cada produção (como em CoCoR).
As funções auxiliares são escritas em C e são incluidas no mesmo ficheiro após a gramática, tal como em LISA.

"Lavanda.y" 23b ≡

```
%{
#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;
}
```

◇

File defined by 23b, 24, 25, 26.

"Lavanda.y" 24 ≡

```
%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;       }
   ;

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;
               }
    ;

◇
```

File defined by 23b, 24, 25, 26.

"Lavanda.y" 25 ≡

```
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;     }
 ;

%%

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);
}
```

◇

File defined by 23b, 24, 25, 26.

"Lavanda.y" 26 ≡

```
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 23b, 24, 25, 26.

Capítulo 6

Implementação LRC

Neste caso a gramática de atributos é especificada em vários ficheiros, como se mostra nas secções seguintes. Há mecanismos de separar a gramática semântica da abstracta. Os atributos são declarados e associados aos símbolos em secção própria e as regras de cálculo são associadas às produções, no fim, escrevendo-as entre '{' e '}' em linguagem própria (SSL).
Devido à existência de vários ficheiros e à forma como devem ser compilados, incluimos numa última secção uma makefile para produzir e invocar o processador.

6.1 Context Free Grammar

"CFG.ssl" 27 ≡

```
/**********************************
   Scanner
**********************************/


WHITESPACE     : < [\ \t\n]+ >;
DM             : < [0-3][0-9] >;
YEAR          : < [0-2][0-9][0-9][0-9] >;
NUMBER          : < [0-9]+ >;
CORPO          : < "corpo" >;
CASA           : < "casa" >;
COR            : < "cor" >;
BR             : < "br" >;
ALG            : < "alg" >;
FIB            : < "fib" >;
LA             : < "la" >;
IDENTIFIER     : < [a-z]+ >;
```

◇

File defined by 27, 28, 29.

"CFG.ssl" 28 ≡

```
/*******************************
   Parser
*******************************/


lavanda  { syn Lavanda ast; };
lavanda  ::= (cabec sacos )
         { $$.ast = RootLavanda( sacos.ast ); }
         ;

cabec   { syn Cabec ast ; };
cabec   ::= (data IDENTIFIER)
        { $$.ast = NoSyn(); }
        ;

data { syn Cabec ast;   };
data ::= (DM '-' DM '-' YEAR)
        { $$.ast = NoSyn(); };

sacos    { syn Sacos ast ; } ;
sacos    ::= ( saco moreSacos )
        { $$.ast = ConsSacos(saco.ast,moreSacos.ast); };


moreSacos { syn Sacos ast; };
moreSacos ::= ()
      { $$.ast = NoSacos();   }
      | ( sacos )
      { $$.ast = sacos.ast;   };

saco { syn Saco ast ; }
     ;
saco ::= (NUMBER IDENTIFIER '(' lotes ')' )
     { $$.ast = ConsSaco(IDENTIFIER,STRtoINT(NUMBER),lotes.ast); }
     ;

lotes { syn Lotes ast;  };
lotes ::= ( lote outros )
     { $$.ast = ConsLotes(lote.ast,outros.ast); };

lote { syn Lote ast; };
lote ::= ( tipo NUMBER )
     { $$.ast = ConsLote(tipo.ast,STRtoINT(NUMBER));  };


outros { syn Lotes ast; };
outros ::= ()
      { $$.ast = NoLotes();   }
      | ( ',' lotes )
      { $$.ast = lotes.ast;   };
```

◇

File defined by 27, 28, 29.

"CFG.ssl" 29 ≡

```
tipo { syn Tipo ast; };
tipo ::= (classe '-' tinto '-' fio )
    { $$.ast = ConsName(classe.ast,tinto.ast,fio.ast);   };

classe { syn Name ast;  };
classe ::= ( CORPO )
      { $$.ast = ConsStr("corpo");  }
     | ( CASA )
      { $$.ast = ConsStr("casa");   };

tinto    { syn Name ast;   };
tinto ::= ( COR )
      { $$.ast = ConsStr("cor"); }
     | ( BR )
      { $$.ast = ConsStr("br");  };

fio { syn Name ast;  };
fio ::= ( FIB )
      { $$.ast = ConsStr("fib"); }
     | ( LA )
      { $$.ast = ConsStr("la");  }
     | ( ALG )
      { $$.ast = ConsStr("alg"); };
```

◇

File defined by 27, 28, 29.

6.2 Abstract Syntax Tree

"AST.ssl" 30 ≡

```
Lavanda ~ lavanda.ast ;
root Lavanda;

Lavanda  : RootLavanda (Sacos)
      ;
Cabec   : NoSyn ()
      ;

Sacos : NoSacos ()
      | ConsSacos  (Saco Sacos)
      ;
Saco  : NoSaco()
      | ConsSaco (STR INT Lotes)
      ;

Lotes : NoLotes()
      | ConsLotes ( Lote Lotes)
      ;

Lote : ConsLote ( Tipo INT)
     ;

Tipo : ConsName(Name Name Name);

Name : ConsStr(STR);


◇
```

6.3 Semantics

"Semantics.ssl" 31 ≡

```
/***************

  Attributes

***************/


/***** b)    *****/

list tableLotes;
tableLotes : Empty()
      | ConsTableLotes ( STR INT tableLotes);


tableLotes: Empty    [ @ ::= "" ]
     | ConsTableLotes   [ @ ::= "\t" @ "\t\t" @ "\n" @ ];



/***** c)    *****/

list tablePrice;
tablePrice: EmptyPrice()
     | ConsTablePrice( STR REAL tablePrice);

tablePrice: EmptyPrice     [ @ ::= ""]
     | ConsTablePrice   [ @ ::= "\t" @ "\t\t" @ "\n\n" @];



list idSacos;
idSacos: EmptyIds()
   | ConsIdSacos (INT idSacos);

idSacos: EmptyIds    [ @ ::= "" ]
   | ConsIdSacos     [ @ ::= "\t" @ "\n\n" @];


/******/

Lavanda { syn INT nSacos;

     syn tableLotes tLotesOut;

     } ;
```

◇

File defined by 31, 32, 33, 34, 35, 36, 37.

"Semantics.ssl" 32 ≡

```
Lavanda : RootLavanda
     {    local STR numSacos;
     local tableLotes tLotes;

          numSacos       = "\n\nNumero de sacos: " # INTtoSTR($$.nSacos) # "\n\n";
          $$.nSacos      = Sacos.nSacos;

     /* b) */
     tLotes         = $$.tLotesOut;
     Sacos.tLotesIn = initEnv(Empty());
          $$.tLotesOut   = Sacos.tLotesOut;

     /* c) */
     Sacos.tPriceIn = initEnvPrice(EmptyPrice());
     Sacos.inIds    = EmptyIds();
     };


Sacos { syn INT nSacos;

   /* b) */
   inh tableLotes tLotesIn;
   syn tableLotes tLotesOut;

   /* c) */
   inh tablePrice tPriceIn;
   inh idSacos inIds;
   syn idSacos outIds;        };

Sacos  : NoSacos
     { $$.nSacos = 0;
       $$.tLotesOut    = $$.tLotesIn;

      /* c) */
      $$.outIds        = $$.inIds;     }

        | ConsSacos
          { $$.nSacos       = Sacos$2.nSacos + 1;

       /* b) */
      Saco.tLotesIn    = $$.tLotesIn;
      Sacos$2.tLotesIn = Saco.tLotesOut;
      $$.tLotesOut     = Sacos$2.tLotesOut;

      /* c) */
      Saco.tPriceIn    = $$.tPriceIn;
      Sacos$2.tPriceIn = $$.tPriceIn;
      Saco.inIds       = $$.inIds;
      Sacos$2.inIds    = Saco.outIds;
      $$.outIds        = Sacos$2.outIds;  };
```

◇

File defined by 31, 32, 33, 34, 35, 36, 37.

"Semantics.ssl" 33 ≡

```
Saco { syn INT nLotes;

   /* b) */

   inh tableLotes tLotesIn;
   syn tableLotes tLotesOut;

   /* c) */
   inh tablePrice tPriceIn;
   syn REAL custoTotal;
   inh idSacos inIds;
   syn idSacos outIds;        };

Saco :   NoSaco
    {
      /* b) */
      $$.nLotes      = 0;
      $$.tLotesOut   = $$.tLotesIn;

      /* c) */
      $$.custoTotal  = 0.0;
      $$.outIds      = $$.inIds;    }
       | ConsSaco
    { local STR infLotesPrice;
      local STR erro;
      infLotesPrice = "\nNumero de lotes no saco n. " #
                           INTtoSTR(INT) # ": " # INTtoSTR($$.nLotes) #
            "\nPreco total: " # REALtoSTR($$.custoTotal);

      $$.nLotes      = Lotes.nLotes;

      /* b)  */
      Lotes.tLotesIn  = $$.tLotesIn;
      $$.tLotesOut    = Lotes.tLotesOut;

      /* c) */
      Lotes.tPriceIn  = $$.tPriceIn;
      $$.custoTotal   = Lotes.custoTotal;
      $$.outIds      = joinId($$.inIds,INT);
      erro            = belongId($$.inIds,INT) ? "Erro: saco ja existente!" : "";
   };

Lotes { syn INT nLotes;

   /* b) */

   inh tableLotes tLotesIn;
   syn tableLotes tLotesOut;

   /* c) */
   inh tablePrice tPriceIn;
   syn REAL custoTotal;

   };

◇
```

File defined by 31, 32, 33, 34, 35, 36, 37.

"Semantics.ssl" 34 ≡

```
Lotes:   NoLotes
    {
      /* a) */
      $$.nLotes = 0;

      /* b) */
      $$.tLotesOut     = $$.tLotesIn;

      /* c) */
      $$.custoTotal    = 0.0;             }
   | ConsLotes
    { $$.nLotes        = Lotes$2.nLotes + 1;

      /* b) */
      Lote.tLotesIn    = $$.tLotesIn;
      Lotes$2.tLotesIn = Lote.tLotesOut;
      $$.tLotesOut     = Lotes$2.tLotesOut;

      /* c) */
      Lote.tPriceIn    = $$.tPriceIn;
      Lotes$2.tPriceIn = $$.tPriceIn;
      $$.custoTotal = Lote.custoTotal + Lotes$2.custoTotal; };


Lote {
   /* b) */
   inh tableLotes tLotesIn;
   syn tableLotes tLotesOut;

   /* c) */
   inh tablePrice tPriceIn;
   syn REAL custoTotal;
     };

Lote:  ConsLote
   {
     /* b) */
     $$.tLotesOut = changeTableLotes($$.tLotesIn,Tipo.name,INT);

    /* c) */
    $$.custoTotal = lookupPrice($$.tPriceIn,Tipo.name) * INTtoREAL(INT);

   };
```

◇

File defined by 31, 32, 33, 34, 35, 36, 37.

"Semantics.ssl" 35 ≡

```
Tipo { syn STR name; };
Tipo : ConsName
   { $$.name = Ident(Name$1) # "-" # Ident(Name$2) # "-" # Ident(Name$3);  };



/*****  b)  **********/

tableLotes initEnv(tableLotes table)
{

   with(table) (
        Empty()                                  : ConsTableLotes("corpo-br-la",0,
                        ConsTableLotes("corpo-br-fib",0,
                        ConsTableLotes("corpo-br-alg",0,
                        ConsTableLotes("corpo-cor-la",0,
                        ConsTableLotes("corpo-cor-fib",0,
                        ConsTableLotes("corpo-cor-alg",0,
                        ConsTableLotes("casa-br-la",0,
                                  ConsTableLotes("casa-br-fib",0,
                        ConsTableLotes("casa-br-alg",0,
                        ConsTableLotes("casa-cor-la",0,
                        ConsTableLotes("casa-cor-fib",0,
                        ConsTableLotes("casa-cor-alg",0,Empty()))))))))))))
            , *               : Empty()
       )
};
◇
```

File defined by 31, 32, 33, 34, 35, 36, 37.

"Semantics.ssl" 36 ≡

```
tableLotes changeTableLotes(tableLotes table,STR name, INT num)
{

   with(table) (
        Empty()                                  : Empty(),
        ConsTableLotes(name1,num1,tail)   : (name1 == name) ? ConsTableLotes(name,num+num1,tail)
                       : ConsTableLotes(name1, num1, changeTableLotes(tail,name,num))
       )
};


STR Ident(Name s)
{
   with(s)
      (ConsStr(s1): s1,
       *      : ""
      )
};


/******** c)   ********/


tablePrice initEnvPrice(tablePrice table)
{

   with(table) (
        EmptyPrice()    : ConsTablePrice("corpo-br-la",1.0,
                ConsTablePrice("corpo-br-fib",2.2,
                ConsTablePrice("corpo-br-alg",3.4,
                ConsTablePrice("corpo-cor-la",4.5,
                ConsTablePrice("corpo-cor-fib",1.9,
                ConsTablePrice("corpo-cor-alg",3.7,
                ConsTablePrice("casa-br-la",2.6,
                ConsTablePrice("casa-br-fib",7.1,
                ConsTablePrice("casa-br-alg",5.3,
                ConsTablePrice("casa-cor-la",3.5,
                ConsTablePrice("casa-cor-fib",2.3,
                ConsTablePrice("casa-cor-alg",2.5,EmptyPrice()))))))))))))
            , *         : EmptyPrice()
       )
};

◇
```

File defined by 31, 32, 33, 34, 35, 36, 37.

"Semantics.ssl" 37 ≡

```
REAL lookupPrice(tablePrice table,STR name)
{

   with(table) (
        EmptyPrice()                          : 0.0,
        ConsTablePrice(name1,num1,tail)   : (name1 == name) ? num1 : lookupPrice(tail,name)
       )
};


idSacos joinId(idSacos ids, INT numSaco)
{
   with(ids)
      (
       EmptyIds()    : ConsIdSacos(numSaco,EmptyIds()),
       ConsIdSacos(num,tail)  : ConsIdSacos(num,joinId(tail,numSaco))
      )
};


BOOL belongId(idSacos ids, INT numSaco)
{
   with(ids)
      (
       EmptyIds()    : false,
       ConsIdSacos(num,tail)  : (num == numSaco) ? true : belongId(tail,numSaco)
      )
};


◇
```

File defined by 31, 32, 33, 34, 35, 36, 37.

6.4 Unparsing

"Unparsing.ssl" 38a ≡

```
Lavanda  : RootLavanda   [ @ ::= @ numSacos
             "Tabela Lotes:\n\t -Descricao- \t -N. Lotes-\n"
             tLotes]
         ;
Cabec    : NoSyn   [ @ ::= ""]
    ;
Sacos    :  ConsSacos    [ @ ::= @ " " @ ]
    ;
Saco     : ConsSaco      [ @ ::= infLotesPrice " " erro]
         ;

◇
```

6.5 Makefile

"makefileLRC" 38b ≡

```
RUNLRC=perl -S -w /usr/local/lrc/bin/runlrc.pl

SSL_SOURCES = LavandaCFG.ssl attributes.ssl

LRC_OPTIONS =  -no_warnings -remove_deadends -v2c_code

eva : code.ivr bcode.ieq
   $(RUNLRC) +4 -4 $(LRC_OPTIONS)

bcode.ieq code.ivr :  $(SSL_SOURCES)
   $(RUNLRC) -3 $(LRC_OPTIONS) $(SSL_SOURCES)

clean :
   $(RUNLRC) -clean
   rm -f eva
   rm -f bcode.*
   rm -f code.*
   rm -f Save.*

◇
```

Capítulo 7

Implementação AntLR

Neste caso voltamos a ter toda a *GA*, incluindo a especificação léxica num único ficheiro — Lavanda.g — como se mostra a seguir. O aspecto geral é semelhante ao CoCoR, em que os atributos são declarados nas produções associados com os respectivos símbolos entre '[' e ']' e as regras de cálculo, escritas também em C#, são delimitadas por '{' e '}' ao longo da produção. As funções auxiliares, em C#, escrevem-se em módulos separados, incluidos na secção 7.2 e 7.3.

7.1 Gramática

"Lavanda.g" 39 ≡

```
header
{
    // gets inserted in the C# source file before any
    // generated namespace declarations
    // hence -- can only be using directives
    using System.Collections;
}

options {

   language  = "CSharp";
   namespace = "Lavanda";          // encapsulate code in this namespace
}

class MyLexer extends Lexer;
options {
   caseSensitive=false;                 // D = d     (on LEXER rules match)
   caseSensitiveLiterals=false;         // FOR = fOr (only for tokens)
}

◇
```

"LexerParser.g" 40 ≡

```
COMMA            : ','   ;
LPAREN            : '('   ;
RPAREN            : ')'   ;
MINUS            : '-'  ;

protected LETTER  : 'a'..'z';
protected DIGIT         : '0'..'9';


IDENTIFIER        : LETTER ( LETTER )* ;
NUMBER            : DIGIT ( DIGIT )* ;

protected NEWLINE
   :
         (
               options { generateAmbigWarnings=false; }
           : '\n'
         | '\r'
           | '\r''\n'
         )
           { newline(); }
   ;

WHITESPACE
   :
      ( ' '
      | '\t'
      | NEWLINE
      )+
      { $setType(Token.SKIP); }
   ;


class MyParser extends Parser;
options {
   //exportVocab=My;
}
{
public OpHashtable opTable;
}

◇
```

File defined by 40, 41, 42, 43.

"LexerParser.g" 41 ≡

```
lavanda
options {defaultErrorHandler=false;}
   {
      int nSacos = 0;
      Hashtable inTable = opTable.initNLotes(),
      outTable,
      inTPrice = opTable.initTablePrice();
      ArrayList inIds = opTable.initIds(),
      outIds = inIds;
   }
      : cabec sacos[out nSacos, inTable, out outTable, inTPrice, inIds, out outIds]
         {
          Console.WriteLine("Total sacos: " + nSacos + "\n");
          opTable.printTableLotes(outTable);
         }
      ;

cabec    : date IDENTIFIER
      ;

date     : NUMBER MINUS NUMBER MINUS NUMBER
      ;

sacos [ out int nSacos , Hashtable inTable, out Hashtable outTable,
     Hashtable inTPrice, ArrayList inIds, out ArrayList outIds]
options {defaultErrorHandler=false;}
   {
      int nSacos1 = 0;
      float custoTotal;
   }
      : saco[inTable, out outTable, inTPrice, out custoTotal, inIds, out outIds]
         { nSacos = 1;}
        ( sacos[out nSacos1, outTable, out outTable, inTPrice, outIds, out outIds] )*
         { nSacos += nSacos1;}
      ;
◇
```

File defined by 40, 41, 42, 43.

"LexerParser.g" 42 ≡

```
saco  [ Hashtable inTable, out Hashtable outTable, Hashtable inTPrice,
     out float custoTotal, ArrayList inIds, out ArrayList outIds]
options {defaultErrorHandler=false;}
   {
      int nLotes = 0, num;
      custoTotal = 0;
   }
      :  t1:NUMBER { num = Convert.ToInt32(t1.getText()); }
         IDENTIFIER
         LPAREN
         lotes[out nLotes, inTable, out outTable, inTPrice, out custoTotal]
         RPAREN { Console.WriteLine("N. lotes no saco " + num + ": " + nLotes);
                opTable.writePrice(custoTotal);
                 if ( inIds.Contains( num ))
                  throw new Exception("Cliente already exists!");
                 outIds = opTable.newId(inIds, num);
               }
      ;

lotes [out int nLotes, Hashtable inTable, out Hashtable outTable,
   Hashtable inTPrice, out float custoTotal]
options {defaultErrorHandler=false;}
   {
      int nLotes1 = 0;
      float custoTotal2 = 0;
   }
      : lote[inTable, out outTable, inTPrice, out custoTotal]
        {nLotes = 1;}
       ( COMMA lotes[out nLotes1, inTable, out outTable, inTPrice, out custoTotal2] )*
        {nLotes += nLotes1; outTable=inTable; custoTotal +=custoTotal2;}
      ;

lote  [Hashtable inTable, out Hashtable outTable, Hashtable inTPrice, out float custoTotal]
options {defaultErrorHandler=false;}
   {
      string name = "";
   }
      : tipo[out name] t1:NUMBER
        {
           outTable = opTable.updateTableLotes(inTable, name, Convert.ToInt32(t1.getText()));
           custoTotal = opTable.lookupPrice(inTPrice, name) * Convert.ToInt32(t1.getText());
        }
      ;
◇
```

File defined by 40, 41, 42, 43.

"LexerParser.g" 43 ≡

```
tipo  [out string name]
options {defaultErrorHandler=false;}
   {
      string name1 = "",
      name2 = "",
      name3 = "";
   }
      : classe[out name1] MINUS tinto[out name2] MINUS fio[out name3]
         { name = name1 + "-" + name2 + "-" + name3; }
      ;

classe   [out string name]
options {defaultErrorHandler=false;}
   {
      name = "";
   }
      : "corpo"   { name = "corpo"; }
      | "casa" { name = "casa"; }
      ;

tinto [out string name]
options {defaultErrorHandler=false;}
   {
      name = "";
   }
      : "br"      { name = "br"; }
      | "cor"     { name = "cor"; }
      ;

fio      [out string name]
options {defaultErrorHandler=false;}
   {
      name = "";
   }
      : "alg"     { name = "alg"; }
      | "la"      { name = "la"; }
      | "fib"     { name = "fib"; }
      ;


◇
```

File defined by 40, 41, 42, 43.

7.2 Funções auxiliares

"Op_Hashtable.cs" 44 ≡

```
using System;
using System.Collections;

namespace Lavanda
{

   public class OpHashtable
   {

      public OpHashtable()
      {
      }

      // alinea b
      public Hashtable initNLotes()
      {
         Hashtable env = new Hashtable();
         env.Add("corpo-br-la",0);
         env.Add("corpo-br-alg",0);
         env.Add("corpo-br-fib",0);
         env.Add("corpo-cor-la",0);
         env.Add("corpo-cor-alg",0);
         env.Add("corpo-cor-fib",0);
         env.Add("casa-br-la",0);
         env.Add("casa-br-alg",0);
         env.Add("casa-br-fib",0);
         env.Add("casa-cor-la",0);
         env.Add("casa-cor-alg",0);
         env.Add("casa-cor-fib",0);

         return env;
      }

      public Hashtable updateTableLotes(Hashtable inTable, String name, int number)
      {
         IDictionaryEnumerator env = inTable.GetEnumerator();

         while(env.MoveNext())
         {
            string key = (string)env.Key;

            if (key.Equals(name))
            {
               int pieces = (int)env.Value + number;
               inTable.Remove(key);
               inTable.Add(key, pieces);
               break;
            }
         }
         return inTable;
      }
```

◇

File defined by 44, 45, 46a.

"Op_Hashtable.cs" 45 ≡

```
public void printTableLotes( Hashtable myList )
{
   IDictionaryEnumerator myEnumerator = myList.GetEnumerator();
   Console.WriteLine( "\t-Descricao-\t-N. Lotes-" );
   while ( myEnumerator.MoveNext() )
      Console.WriteLine("\t{0}:\t{1}", myEnumerator.Key, myEnumerator.Value);
   Console.WriteLine();
}



// alinea c

public Hashtable initTablePrice()
{
   Hashtable env = new Hashtable();
   env.Add("corpo-br-la",1.0f);
   env.Add("corpo-br-alg",2.2f);
   env.Add("corpo-br-fib",3.4f);
   env.Add("corpo-cor-la",4.5f);
   env.Add("corpo-cor-alg",3.7f);
   env.Add("corpo-cor-fib",1.9f);
   env.Add("casa-br-la",2.6f);
   env.Add("casa-br-alg",5.3f);
   env.Add("casa-br-fib",7.1f);
   env.Add("casa-cor-la",3.5f);
   env.Add("casa-cor-alg",2.5f);
   env.Add("casa-cor-fib",2.3f);

   return env;
}



public float lookupPrice (Hashtable inPrice, string name )
{
   float price = 0;

   IDictionaryEnumerator env = inPrice.GetEnumerator();
   while ( env.MoveNext() )
   {
      if (env.Key.Equals(name))
      {
         price = (float)env.Value;
         break;
      }
   }

   return price;
}
```

◇

File defined by 44, 45, 46a.

"Op_Hashtable.cs" 46a ≡

```
            public ArrayList initIds()
            {
               return ( new ArrayList() );
            }


            public ArrayList newId(ArrayList old, int num)
            {
               old.Add(num);
               return ( (ArrayList)old.Clone() );
            }

            public void writePrice(float num)
            {
               Console.WriteLine("Preco Total: " + num + "\n" );
            }
         }
      } // end namespace

      ◇
```

File defined by 44, 45, 46a.

7.3 Main

"LavandaCompiler.cs" 46b ≡

```
      using System;
      using System.Collections;
      using System.IO;
      using antlr;
      using Lavanda;

      public class LavandaCompiler {

         public static void Main (string[] args)
         {
            if (args.Length > 0) {
                     Stream stream = new FileStream(args[0], FileMode.Open,
                        FileAccess.Read, FileShare.Read);
                     MyLexer lexer = new MyLexer(stream);
                     MyParser parser = new MyParser(lexer);
                     parser.opTable = new OpHashtable();
            parser.lavanda();


            } else
               Console.WriteLine("-- No source file specified");
         }



      }
      ◇
```

Capítulo 8

Resultados execução

Consideremos como teste o seguinte exemplo:

"Teste.test" 47 ≡

```
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)
```

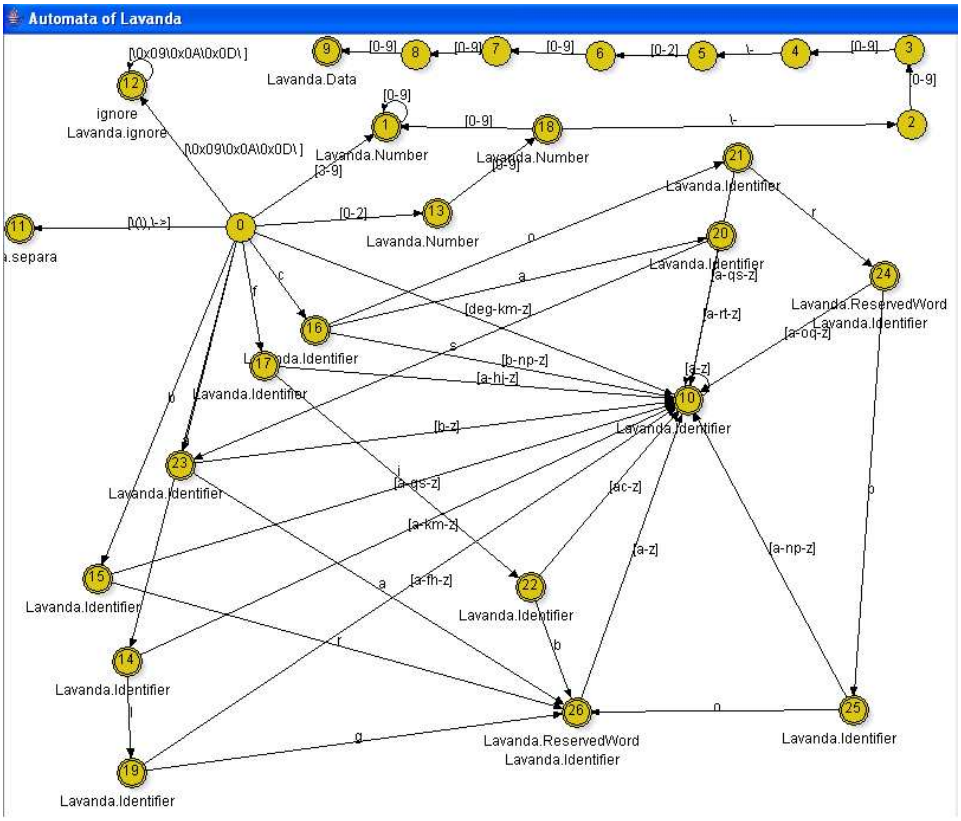
◇

8.1 Resultados obtidos no LISA

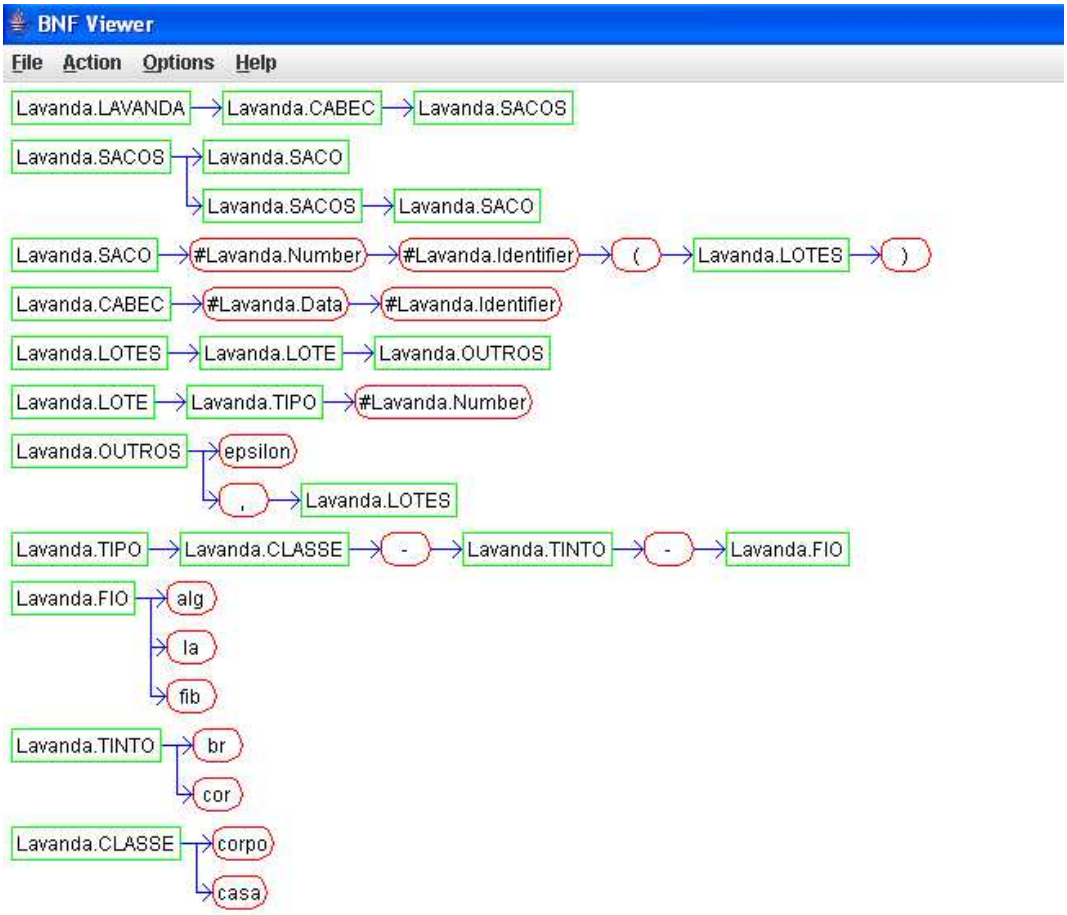
Antes de mostrarmos os resultados em termos de execução do exemplo acima referido poderemos mostrar algumas das funcionalidades suportadas pelo LISA que contribui de forma significativa para a compreensão da compilação de um programa.

Algumas dessas funcionalidades são:

1. `Autómato:` O autómato produzido pelo LISA para a linguagem *Lavanda* é o seguinte:


Automata of Lavanda
Lavanda.Data
ignore
Lavanda.ignore
Lavanda.Number
Lavanda.Number
Lavanda.Number
Lavanda.Identifier
Lavanda.Identifier
Lavanda.Identifier
Lavanda.Identifier
Lavanda.Identifier
Lavanda.Identifier
Lavanda.Identifier
Lavanda.Identifier
Lavanda.Identifier
Lavanda.ReservedWord
Lavanda.Identifier
Lavanda.ReservedWord
Lavanda.Identifier
Lavanda.Identifier
Lavanda.Identifier
Lavanda.Identifier

2. BNF



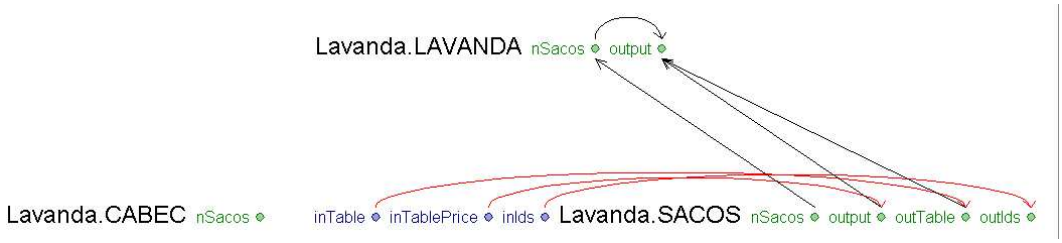
3. `Atributos herdados e sintetizados:`

   Vejamos, por exemplo, a árvore dos atributos herdados e sintetizados para algumas das produções da nossa *Gabs*.

   Para a produção:

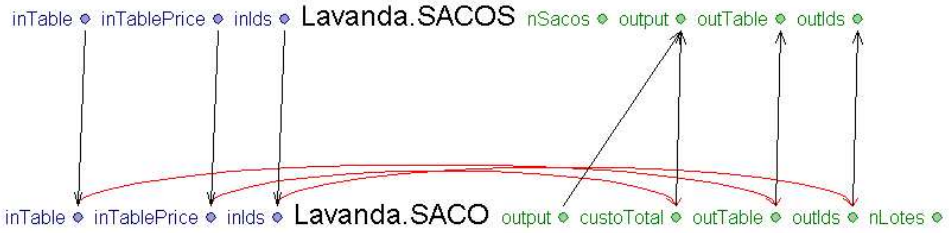
   ```
   p1a:  Lavanda  -->  Cabec  Sacos
   ```

   Os atributos herdados e sintetizados:



   Para a produção:

   ```
   p3a:  Sacos    -->  Saco
   ```



4. `Firs/Follow`

   O LISA permite-nos ainda calcular o first e o follow das produções da *Gabs*.

```
FOLLOW(Lavanda.CABEC)=[#Lavanda$Number]
FIRST(Lavanda.CABEC ::= #Lavanda.Data #Lavanda.Identifier)=[#Lavanda$Data]
FOLLOW(Lavanda.CLASSE)=[-]
FIRST(Lavanda.CLASSE ::= corpo)=[corpo]
FIRST(Lavanda.CLASSE ::= casa)=[casa]
FOLLOW(Lavanda.FIO)=[#Lavanda$Number]
FIRST(Lavanda.FIO ::= alg)=[alg]
FIRST(Lavanda.FIO ::= la)=[la]
FIRST(Lavanda.FIO ::= fib)=[fib]
FOLLOW(Lavanda.LAVANDA)=[]
FIRST(Lavanda.LAVANDA ::= Lavanda.CABEC Lavanda.SACOS)=[#Lavanda$Data]
FOLLOW(Lavanda.LOTE)=[), ,]
FIRST(Lavanda.LOTE ::= Lavanda.TIPO #Lavanda.Number)=[corpo, casa]
FOLLOW(Lavanda.LOTES)=[)]
FIRST(Lavanda.LOTES ::= Lavanda.LOTE Lavanda.OUTROS)=[corpo, casa]
FOLLOW(Lavanda.OUTROS)=[)]
FIRST(Lavanda.OUTROS ::= epsilon)=[epsilon]
FIRST(Lavanda.OUTROS ::= , Lavanda.LOTES)=[,]
FOLLOW(Lavanda.SACO)=[#Lavanda$Number]
FIRST(Lavanda.SACO ::= #Lavanda.Number #Lavanda.Identifier ( Lavanda.LOTES ))=[#Lavanda$Number]
FOLLOW(Lavanda.SACOS)=[#Lavanda$Number]
FIRST(Lavanda.SACOS ::= Lavanda.SACO)=[#Lavanda$Number]
FIRST(Lavanda.SACOS ::= Lavanda.SACOS Lavanda.SACO)=[#Lavanda$Number]
FOLLOW(Lavanda.TINTO)=[-]
FIRST(Lavanda.TINTO ::= br)=[br]
FIRST(Lavanda.TINTO ::= cor)=[cor]
FOLLOW(Lavanda.TIPO)=[#Lavanda$Number]
FIRST(Lavanda.TIPO ::= Lavanda.CLASSE - Lavanda.TINTO - Lavanda.FIO)=[corpo, casa]
```

Assim, depois da análise destas funcionalidades poderemos então mostrar o *output* produzido pelo compilador:

```
Generated compiler/interpreter
Switching to default
Parsing
File parsed in 0.0 s.
Evaluating
Lavanda.LAVANDA
nSacos:3:true
output:

SACOS:

Numero de sacos: 3

LOTES:

Numero de lotes no saco 1 : 2
Preco Total: 9.5


Numero de lotes no saco 2 : 1
Preco Total: 28.4


Numero de lotes no saco 3 : 6
Preco Total: 40.6

TABELA LOTES:

        -Descricao-     -Lotes-
        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-cor-fib:    1
        corpo-br-alg:     0
        casa-cor-fib:     1
:true

Program evaluated in 0.01 s.

C:\lisa\java>_
```

8.2 Resultados obtidos no CoCoR

Vejamos o resultado produzido pelo compilador para a linguagem **Lavanda** usando o CoCoR.

```
Nº lotes no saco 1: 2
Preco Total: 9.5

Nº lotes no saco 2: 1
Preco Total: 28.4

Nº lotes no saco 3: 6
Preco Total: 40.6

Total sacos: 3

        -Descricao-     -Nº Lotes-
        casa-cor-alg:   4
        casa-br-la:     0
        corpo-br-la:    0
        corpo-br-fib:   0
        corpo-cor-la:   4
        corpo-cor-fib:  1
        casa-cor-la:    3
        corpo-br-alg:   0
        casa-br-fib:    4
        corpo-cor-alg:  2
        casa-br-alg:    0
        casa-cor-fib:   1
```

8.3 Resultados obtidos no **Lex/Yacc**

```
[_localhost Lex-Yacc]# ./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
```

8.4 Resultados obtidos no LRC

```
[_localhost LavandaLRC]$ ./eva -O Teste.test
entering output descriptor: attr='(null)' view='BASEVIEW' file='(null)'
/**** processing file Teste.test ****/
Unparse decorated term

Numero de lotes no saco n. 1: 2
Preco total: 9.5
Numero de lotes no saco n. 2: 1
Preco total: 28.4
Numero de lotes no saco n. 3: 6
Preco total: 40.6

Numero de sacos: 3

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

Resultados obtidos no **AntLR**

Capítulo 9

Ficheiros no Documento