Questões em Tradução de Linguagens

Capítulos 3 do Pratt e do Sebesta

- Sintaxe de uma LP é a forma de suas expressões, instruções e unidades de programa:
  - descreve a seqüência de símbolos que tornam válido um programa,
  - em uma linguagem natural é o arranjo de palavras como elementos em uma seqüência para mostrar o relacionamento entre elas.

  $\Rightarrow$ sintaxe é o conjunto de regras que determinam quando uma sentença é bem formada.
- Semântica de uma LP é o significado de suas expressões, instruções e unidades de programa.

Introdução

- Quem deve usar as definições de uma LP:
  - outros projetistas de linguagens,
  - implementadores,
  - programadores (usuários da linguagem)
- definições básicas:
  - sentença: string de caracteres sobre um alfabeto.
  - linguagem: conjunto de sentenças.
  - lexema: unidade de menor nível sintático de uma linguagem (p.ex.: *, for, begin).
  - token: categoria de lexemas (p.ex.: identificador).
  - símbolos (tokens) são combinados em sentenças para formar um programa.

Questões em Tradução de Linguagens

Regras sintáticas são suficientes?

- Anos 60: para projetar LP só era necessário uma sintaxe formal, tipo BNF (Backus-Naur Form).

  <inteiro> ::= <sinal><digito>|<inteiro><digito>

  – Em expressões ambíguas, as regras sintáticas não são suficientes para determinar as interpretações corretas:
    - associativa: 1-2-3 = -4 ou 2 ?
    - precedência: 2+3*4 = 20 ou 14 ?
    - tipos: x = 2.45+3.67 = 5, se x e + são inteiros
      = 6, se x inteiro e + real
      = 6.12, se x e + são reais

- Declarações, seqüência de controle, operações, ambiente de referência, etc, são necessários.

Sintaxe de LP

Critérios gerais sintáticos

- Propósito principal da sintaxe:
  - prover notação para comunicação entre o programador e o processador da linguagem.
- Propósitos secundários da sintaxe:
  - legibilidade: ler e entender facilmente,
  - capacidade de escrita: facilitar a programação,
  - facilidade de verificação: facilitar o exame da exatidão do código,
  - facilidade de tradução: facilitar trabalho do tradutor,
  - ausência de ambigüidade.

Critérios Gerais de Sintaxe

- Legibilidade
  - Pela inspeção do programa, as estruturas dos algoritmos e dos dados DEVEM ficar aparentes, sem necessitar consultar documentação adicional:
    - cmd estruturados palavras chaves comentários
      declarar dados op. mnemônicos campo-livre
      variedade de construções sintáticas

Exemplo: Cobol (auto-documentado)
Contra-exemplo: programas Lisp ou Mumps

Legibilidade

Programa Cobol (exemplo)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SUM-OF-PRICES.
AUTHOR. T-PRATT.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. SUN.
OBJECT-COMPUTER. SUN.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT ARQ-ENTRADA  ASSIGN TO INPUT.
    SELECT ARQ-SAIDA  ASSIGN TO OUTPUT.
DATA DIVISION.
FILE SECTION.
FD ARQ-ENTRADA  LABEL RECORD IS
     OMITTED.
01 ITEM-PRECO.
       02  ITEM     PICTURE   X(30).
       02  PRECO  PIC          9999V99.
FD ARQ-SAIDA.
 01 LINHA-IMPRESSORA  PIC X(80).
WORKING-STORAGE  SECTION.
77 TOTAL          PIC 9999V99, VALUE 0,
     USAGE IS COMPUTATIONAL.
77 CONTADOR   PIC  9999, VALUE 0, USAGE IS
     COMPUTATIONAL.
01 LINHA-TOTAL.
     02 FILLER VALUE  'SOMA  = ' PIC X(12).
     02 TOTAL-SOMA PIC   $$,$$$,$$9.99.
     02 FILLER   PIC  X(15), VALUE  '  TOTAL DE
     ITENS   '.
     02 TOTAL-CONTADOR  PIC  ZZZ9.
01 LINHA-SAIDA.
    02 CONTADOR  PIC  Z,ZZ9.
    02 ITEM            PIC X(30).
    02 FILLER          PIC X(05), VALUE SPACES.
    02 PRECO           PIC $,$$9.99.
```

Legibilidade

Programa Cobol (continuação do exemplo)

```
PROCEDURE DIVISION.
INICIA.
        OPEN INPUT ARQ-ENTRADA  AND OUTPUT ARQ-SAIDA.
LEDADOS.
         READ ARQ-ENTRADA AT END GO TO FINALIZA.
         ADD PRECO  OF ITEM-PRECO TO  TOTAL.
         ADD 1 TO  CONTADOR.
         MOVE CORRESPONDING  ITEM-PRECO     TO   LINHA-SAIDA.
         MOVE CONTADOR           TO  CONTADOR  OF  LINHA-SAIDA.
         WRITE LINHA-IMPRESSORA FROM LINHA-SAIDA.
         GO TO LEDADOS.
FINALIZA.
        MOVE   TOTAL  TO   TOTAL-SOMA.
        MOVE   CONTADOR TOTAL-CONTADOR.
        WRITE  LINHA-IMPRESSORA FROM LINHA-TOTAL.
        CLOSE  ARQ-ENTRADA  AND  ARQ-SAIDA.
        STOP RUN.
```

Legibilidade

Programa Lisp (contra-exemplo)

```
;; ARQ-ENTRADA COM 0 OU MAIS REGISTROS (LISTA), CADA QUAL CONTENDO ITEM E PRECO.
;; LE ARQ-ENTRADA E GRAVA SEUS REGISTROS NO ARQ-SAIDA, TOTALIZANDO OS VALORES.
;; ULTIMO REG: TOTAL DE REGISTROS GRAVADOS E TOTAL DE VALORES ACUMULADOS
;;

(DEFUN PROCEDIMENTO(ARQ-ENTRADA  ARQ-SAIDA)

   ((PROBE-FILE ARQ-ENTRADA)
        (OPEN-INPUT-FILE       ARQ-ENTRADA)
         (OPEN-OUTPUT-FILE   ARQ-SAIDA)
         (SETQ CONTADOR  0   TOTAL  0)
          (LOOP
              (TERPRI)
              ((NULL (SETQ ITEM-PRECO (READ)))        ;; EOF
                    (PRINT  (PACK*  "QTE DE ITENS =  " CONTADOR "     VALOR TOTAL = " TOTAL)) )
              (INCQ CONTADOR 1)                       ;;  INCREMENTA CONTADOR ITENS LIDOS
              (INCQ TOTAL (NTH 1 ITEM-PRECO))         ;; ADICIONA O VALOR DO ITEM AO TOTAL
              (PRINT (PACK* "ORDEM " CONTADOR   "   ITEM= " (CAR ITEM-PRECO)
                            "    VALOR =  "  (CDR ITEM-PRECO)))  )
          (CLOSE-OUTPUT-FILE   ARQ-SAIDA)
          (CLOSE-INPUT-FILE      ARQ-ENTRADA) ))
```

Critérios Gerais de Sintaxe

- Facilidade de escrever
  - Características sintáticas que facilitam escrever um programa:
    - Enfatizar estruturas sintáticas concisas e regulares

      i++;

      for (<exp1>;<exp2>;<exp3>) <comando>;
    - Convenções sintáticas implícitas
      - declaração implícita de inteiro (I-N) e real em Fortran.
      - regras de associatividade e prioridade de avaliação implícitas de operadores (+-/*)
    - Comandos estruturados, operadores mnemônicos, campo-livre, tamanho livre de identificadores, etc.

  Exemplo: Fortran, C, C++

  Contra-exemplo: Pascal, Cobol

Facilidade de escrever

Programa Fortran

```
      PROGRAM MAIN
C     OPERACAO: DADO DUAS MATRIZES QUADRADAS A E B, OBTER  C = A + B
      PARAMETER (MAX=99)
      INTEGER T
      REAL  A (MAX,MAX), B (MAX,MAX), C (MAX,MAX)
 10   WRITE(6, 100)  MAX
 100  FORMAT(" ENTRE COM A DIMENSAO DAS MATRIZES. O MAXIMO EH = ", I5)
      READ (5, 200) T
 200  FORMAT(I5)
      IF (T.LE.0.OR.T.GT.MAX) GO TO 500
      PRINT *, "ENTRE COM OS VALORES  DA  MATRIZ  A"
      READ *, (A(L,K) , L=1,T, K=1,T)
      PRINT *, "ENTRE COM OS VALORES  DA  MATRIZ  B"
      READ *, (B(L,K),L=1,T, K=1,T)
      DO 400 K=1,T
         DO 300 L=1,T
 300        C(L,K) = A(L,K) + B(L,K)
 400  CONTINUE
      PRINT *,  (C(L,K), L=1,T,K=1,T)
      GO TO 800
 500  WRITE(6, 600) MAX
 600  FORMAT( "DIMENSAO ERRADA. MENOR QUE ZERO OU MAIOR QUE ", I5)
      GO TO 10
 800  STOP
      END
```

Facilidade de escrever

Programa Pascal (contra exemplo)

```
Program somatrizes (input,output,infile);
  const max=99;
  type mat_real = array [1..max, 1..max]
          of real;
  var infile: text;   a,b,c: mat_real;
          l,k,t: integer;
  begin
      writeln ('Entre com a dimensão
          das matrizes quadradas. Valor
          máximo é ', max:5);
      repeat   readln (t);
           if (t <=0) or (t>max)
          writeln ('valor da dimensão
                      invalido');
      until (t >0) and (t < max);
      writeln ('entre com o valores da
          matriz A, por linha');
      for l:=1 to t do        { lê a matriz A}
          for k:=1 to t do  read (a[l,k]) ;
      for l:=1 to t do        {lê a matriz B}
          for k:=1 to t do
              begin read (b[l,k]) ;
                    c[l,k] := a[l,k]+b[l,k]
              end;
      for l:=1 to k do
          begin writeln;
                for k:=1 to k do
                    write (c[l,k]:10:2);
           end;
end. { fim do programa}
```

Critérios Gerais de Sintaxe

- Facilidade de Verificação
  - As estruturas sintáticas da LP devem facilitar o exame da exatidão do código gerado:
    - envolve aspectos sintáticos e semânticos.
    - entender automaticamente cada comando é fácil.
    - o processo de criar um programa correto é extremamente difícil.
    - há necessidade de técnicas para provar matematicamente a corretude de um programa.

Exemplo: linguagens declarativas puras.
Contra-exemplo: linguagens imperativas.

Critérios Gerais de Sintaxe

- Facilidade de tradução
  - A construção de tradutores é facilitada pela:
    - regularidade das estruturas sintáticas.
    - pequena variedade de estruturas.

  Exemplos: Lisp, Haskell, Hugs, ML

  pela simplicidade de suas estruturas são ruins de ler e escrever mais fáceis de traduzir.

  Contra-exemplos: Cobol

  semântica simples, fácil de ler, ruim para escrever e difícil de traduzir devido a variedade de estruturas (comandos e declarações).

Critérios Gerais de Sintaxe

- Ausência de ambigüidade
  - Idealmente, cada construção sintática deve ter uma única interpretação:
    - nem sempre acontece nas LP, pois uma estrutura ambígua permite duas ou mais interpretações diferentes.
  - A interpretação de uma estrutura sintática isoladamente não traz problemas.
  - A ambigüidade aparece quando são consideradas combinações entre as diversas estruturas sintáticas permitidas pelas regras da LP.

Critérios Gerais de Sintaxe

- Ausência de ambigüidade
  Exemplos:
  a) chamada de funções e referência a arrays em Fortran:
     A(I,J) é chamada de função ou referência ao elemento a_{ij} do array A?
  b) aninhamento de if
     if <bexp> then <comando$_1$> else <comando$_2$> (ok)
     if <bexp> then <comando$_1$> (ok)
     if <be$_1$> then if <be$_2$> then <comando$_1$> else comando$_2$; (?)
     - em Algol? R.: uso de begin e end.
     - em C e Pascal? R.: regra arbitrária: else se refere ao then mais próximo.
     - em Ada? R.: uso de endif.

Elementos Sintáticos de uma LP

- Conjunto de caracteres
  - Ao projetar a sintaxe de uma LP, a primeira escolha é o conjunto de caracteres. (tendência é usar Unicode?)
- Identificadores
  - Uma cadeia de letras e dígitos, começando com uma letra, é largamente aceito.
- Símbolos de operadores
  - Usual adotar uma combinação de caracteres especiais para alguns operadores e identificadores para outros.
- Palavra-chave
  - identificador usado como uma parte fixa de um comando.
  - Palavra-reservada não pode ser usada pelo programador.
  - Palavras opcionais (noise): melhorar a legibilidade.

Elementos Sintáticos de uma LP

- Comentários
  - Texto inserido no programa com propósito de documenta-lo. É tratado ao nível do analisador léxico.
    - Linha de comentário (toda a linha), com campo fixo.

      Fortran C (na coluna 1) seguido do comentário.
    - Delimitado por caracteres especiais (mais de uma linha).

      C /* comentário qualquer, sem limites de linhas */

      Pascal (* comentário *) ou { comentário }
    - Inicia em qualquer posição, indo até o final da linha.

      LISP ; seguido do comentário

      ADA - seguido do comentário

      C++ // seguido do comentário

      Fortran 90 ! seguido do comentário
- Espaço em branco (tem papel sintático)
  - Usado como separador, exceto se em uma string.

Elementos Sintáticos de uma LP

- Exemplos de uso de espaço em branco
  - Cobol: move x to y ≡ move x to y
  - Pascal
    a) while b < c do b:=b+1;
    b) st := `João e Maria` + ` ` + `são casados`;
  - Lisp
    (defun mdc(a b)(cond ((= b 0) a) (T (mdc b (mod a b)))))
- Delimitador e agrupamento (brackets)
  - Marca o início ou fim de unidade sintática
    comando, expressão, etc
  - Aumentam a legibilidade, facilitam a análise sintática e removem ambigüidades.
  - Chaveamento (brackets) são pares de delimitadores.
    (...), begin...end

Elementos Sintáticos de uma LP

Campos de formato fixos ou livres

- Sintaxe de campo fixo (estritamente)
  Cada elemento do comando precisa aparecer numa dada posição da linha de entrada.
  Ex.: versões antigas da linguagem Assembler, JCL, etc.
- Sintaxe de campo fixo (parcialmente)
  Alguns elementos do comando tem posição fixa outros são livres.
  Ex.: Fortran, Cobol
- Sintaxe de campo livre
  Os elementos do comando podem começar em qualquer lugar da linha de entrada e os elementos na seqüência podem ser separados por um ou mais espaços.
  Ex.: Algol, Pascal, C, etc.

Elementos Sintáticos de uma LP

Expressões

- São funções que acessam os objetos de dados e retornam algum valor.
- São os blocos básico de construções de comandos.
- Em linguagens imperativas, em combinação com o comando de atribuição, permitem alterar o estado da máquina. Ex.: A := cos(x)+y^2;
- Em linguagens funcionais, o fluxo de controle é feito pela avaliação de expressões (funções).
  Ex. Em Lisp, um programa é uma expressão simbólica
  (mapcar ‘(lambda(x) (* x x)) ‘(1 2 3 4 5 6 7 8))

Elementos Sintáticos de uma LP

Comando

- É o principal elemento sintático das linguagens imperativas.
- As linguagens funcionais puras não possuem comandos; elas são declarativas.
- Os comandos podem ser simples ou compostos (estruturados ou aninhados)
- A sintaxe dos comandos influi na ortogonalidade, legibilidade e facilidade de escrita de uma linguagem.
- Cobol tem uma sintaxe de comandos prolixa, muito específica para cada tipo de comando.

Estrutura do Programa em Subprogramas

- Definição de subprogramas separados (Fortran)
- Definição de dados separados (Java)
- Definição de subprogramas aninhados (Pascal)
- Definição separada de interfaces (C)
- Definição de subprogramas sem programa Principal (Lisp, linguagens declarativas)

Estrutura do Programa em Subprogramas

Definição de subprogramas separados

- Unidade sintática distinta do programa, compilada em separado e ligada ao programa em tempo de carga.
- Requer declarar todos os dados do subprograma (de forma explícita ou implícita), inclusive os partilhados.

Ex.: Fortran

Estrutura do programa em Subprogramas

Definição de dados separados

- Agrupa as operações que manipulam um objeto.
- Implementa mecanismo de classes e herança.
- Próprio de linguagens orientadas a objeto.

Ex.: Java, C++, Smalltalk

Estrutura do programa em Subprogramas

Definição de subprogramas aninhados

- Subprogramas são declarados dentro do programa que os utiliza.
- Um subprograma pode declarar outros subprogramas, internos a ele.
- Permite checagem de tipo estática para referências não locais (variáveis externas).

Ex.: Pascal e Algol

Estrutura do programa em Subprogramas

Definição separada de interfaces

- Módulos ou pacotes (packages) de interfaces (arquivos tipo .h).
- Módulos ou pacotes de implementação (arquivos tipo .c).
- Módulos compilados podem ser ligados para criar um programa executável.

C, ML e Ada suportam definição separada de interfaces.

Estrutura do programa em Subprogramas

Definição separada de interfaces

```
/* arquivo pilha.h  */
/* arquivos header (include) exportam
     declarações para clientes */
typedef   struct pilha
    { int elementos[100];   /* pilha de 100
                                 inteiros  */
      int topo = 0;       /* inicia com zero */
     } ;
void       empilha (pilha, int) ;
int        desempilha (pilha) ;
/******   fim de arquivo   *********/

/* arquivo pilha.c *
/* implementa as operações da pilha */

#include “pilha.h”
void       empilha(pilha s, int i)   {
     s.elementos[s.topo++] = i; }
```

```
int desempilha (pilha s) {
     return s.elementos[--s.topo];
}
/****** fim do arquivo *****/

/* um cliente de pilha */
#include “pilha.h”
void       main()
  { pilha s1, s2;    /* declara duas
                        pilhas */
     int    i;
     empilha(s1, 5);
     empilha(s2, 6);
     . . .
     i = desempilha(s1) ;
     . . .   }
```

Estrutura do Programa em Subprogramas

Subprogramas sem programa principal

- **Um arquivo é uma seqüência de expressões.**
- **Um subprograma é declarado em um arquivo.**
- **Um subprograma pode declarar localmente dados e outros subprogramas.**
- **Um arquivo pode conter vários subprogramas.**
- **Um subprograma pode invocar qualquer número de subprogramas, declarados em qualquer arquivo.**
- **Um arquivo pode conter variáveis fora de funções (variáveis globais).**
- **Qualquer variável usada em uma função que não lhe é local, é global.**
- **Os arquivos são carregados para dentro de um ambiente de execução.**
- **Em tempo de carga (load), funções e variáveis ficam definidas.**
- **Uma função pode ser invocada e executada diretamente no ambiente.**

Estrutura do programa em Subprogramas

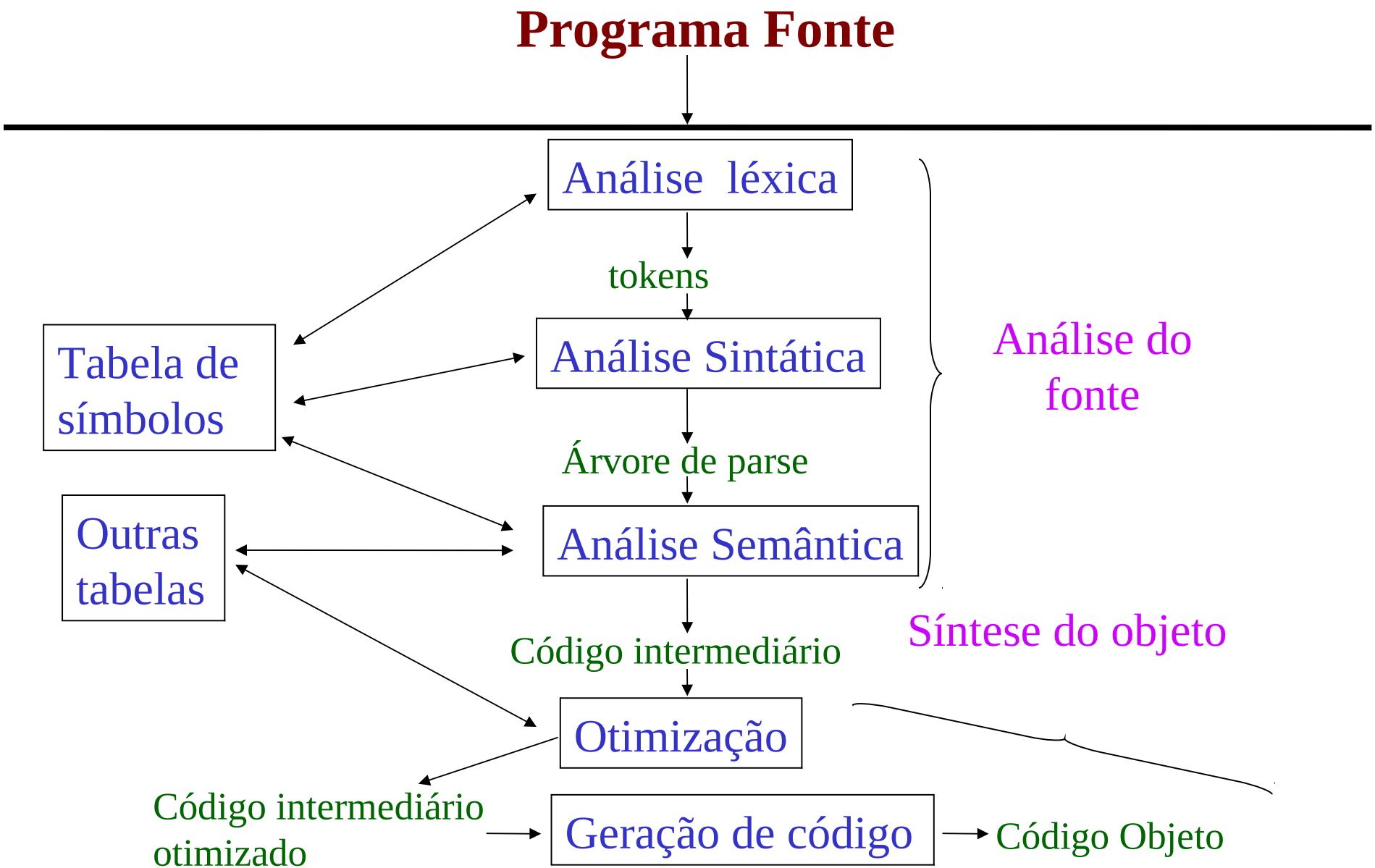
Subprogramas sem programa principal

```
;; arquivo takedrop.lsp
(defun takedrop(n xs)
  (cond
   ((or (= n 0) (null xs)) (list nil xs)
    (T (let  ( (td (takedrop (- n 1) (cdr xs)))
               (y (car td))    (z (cdr td)) )
          (list (cons (car xs) y) z)  ))))
* * * * * * * * * * * * * * * * * * * * *
;; arquivo mergesort.lsp
(load takedrop.lsp)
(defun merge(xs ys comp)
 (progn
   (cond
     ((null xs) ys)
     ((null ys) xs)
     ((apply comp (setq x (car xs)) (setq y
    (car ys)) )
          (cons x (merge (cdr xs) ys comp))  )
     (T  (cons y (merge xs (cdr ys)
    comp))  ))))
```

```
(defun mergesort (xs comp) ; classifica lista p/ comp
   (cond
       ((nul (cadr xs)) xs) ;; tamanho de xs é menor
     que 2
       (T  (let  (  (n (truncate (length xs) 2)
                 (rs (takedrop  n xs)) ( ys (car rs))
                   (zs (cadr rs))  )
           (merge (mergesort ys comp) (mergesort zs
     comp) comp) ))))
(defun fib(n)
  (progn  (defun fibx (a b n)
             (cond ((= n 0)  a)
                   (T (fibx b (+ a  b)  (- n 1))) ))
          (fibx 1 1 n) ))
;; ======== Ambiente lisp
 $ (load mergesort.lsp)
 TAKEDROP  MERGE  MERGESORT  FIB
$ (mergesort '(5 4 3 2 1)  '<)
(1 2 3 4 5)
$ (fib 5)
      8
```

Estágios de Tradução

- Divisão Lógica
  - **Análise** de todo o programa fonte e **síntese** do programa executável após o término da análise.
  - Em muitas implementações de tradutores a análise e síntese se alternam, freqüentemente com base no tratamento comando-a-comando.
- Número de passos
  Grosseiramente, os tradutores são classificados pelo número de passos:
  - leituras do arquivo fonte original e dos demais arquivos gerados a partir deste.


Programa Fonte
Análise léxica
tokens
Análise Sintática
Árvore de parse
Análise Semântica
Código intermediário
Otimização
Código intermediário otimizado
Geração de código
Código Objeto
Tabela de símbolos
Outras tabelas
Análise do fonte
Síntese do objeto

Análise do Fonte

Análise léxica (scanner)

- Grupa seqüência de caracteres em constituintes elementares do fonte (símbolos):
  - identificadores, delimitadores, operadores, números, palavras chaves, palavras opcionais, comentários, ...
- Identifica os itens léxicos (tokens) e os repassa aos outros estágios do tradutor.
- Identifica o tipo de cada token e lhe anexa um rótulo de tipo.
- Converte números para sua representação binária (inteiro e reais).
- Inclui identificadores na TABELA DE SÍMBOLOS e substitui referências ao identificador pelo endereço dado pelo semântico.
- Em geral o scanner (analisador léxico) é modelado como um autômato de estados finitos.

Análise do Fonte

Análise sintática (parsing)

- Identifica as unidades sintáticas do programa:
  (comandos, declarações, expressões, etc) usando os itens léxicos.
- Após identificar uma unidade sintática, chama o analisador semântico para processar essa unidade.
- O analisador sintático coloca os diversos elementos da unidade sintática em uma pilha. A seguir eles são recuperados e processados pelo analisador semântico.
- Há muito esforço na busca de técnicas de análise eficientes, sobretudo aquelas baseadas em gramáticas formais (tais como a BNF).

Análise do Fonte

Análise semântica

- Fase crucial: processa a unidade sintática e começa a gerar a estrutura do código objeto.
- É uma ponte entre as fases de análise e síntese.
- Mantém a tabela de símbolos, detecta a maioria dos erros e, se não houver pré-passo, expande macros e executa diretivas (executáveis em tempo) de compilação.
- Em traduções simples, pode gerar o código objeto.
- Em geral, gera um forma interna de código objeto que passa por um estágio de otimização do tradutor antes de ser gerado o código objeto final.

Análise do Fonte

Análise semântica (continuação)

- O analisador semântico = {analisador 1,...,analisador k}
  O analisador i manipula um tipo particular de unidade sintática do programa, a unidade i, i=1,k.
- Os analisadores interagem via informações armazenadas em várias estruturas de dados, em especial a TS.

Ex.: o as1 trata apenas declarações e as2 trata apenas expressões.
x= y*1.23 + 52; Se x é real e y inteiro, então y e 52 podem ser promovidos para reais, os operadores * e + também podem ser considerados reais e a precedência dos operadores explicitadas.

 As funções do analisador semântico variam muito e dependem da LP e da organização lógica do tradutor.

Síntese do Programa objeto

- Gera o código executável a partir da saída do analisador semântico.
- Pode incluir otimização do código, com base em algoritmos bem conhecidos.
- O uso de subprogramas traduzidos em separado, ou de biblioteca de subprogramas exige o estágio de ligação e carga (mas não necessariamente a inclusão de código externo, p. ex. código de .dll)

Síntese do Programa objeto

Otimização

Expressão

A = B + C + D

Código intermediário (semântico)

a) T1 = B + C

b) T2 = T1 + D

c) A = T2

Código direto e ineficiente gerado

```
1. MOV AX,B    ; op destino,origem
2. ADD AX,C
3. MOV T1,AX
4. MOV AX,T1
5. ADD AX,D
6. MOV T2,AX
7. MOV AX,T2
8. MOV A,AX
```

- **Código otimizado**

```
1. MOV AX,B
2. ADD AX,C
3. ADD AX,D
4. MOV A,AX
```

- **Muitos compiladores usam recursos sofisticados para otimizar:**
  - **cálculos de valores comuns**
  - **eliminação de atribuições constantes em loops**
  - **cálculo de subscritos**
  - **variáveis temporárias, etc.**

Síntese do Programa objeto
Geração de código

- **Após o programa ter sido otimizado é gerado código:**

  - linguagem de máquina real ou
  - assembly ou
  - linguagem de máquina para um computador virtual.

- **O código de saída pode ser**
  - diretamente executado ou
  - montado ou
  - ligado e carregado (requer código relocável!)

Modelos Formais para Tradução

Gramática

- Consiste num conjunto de regras (*produções*) que definem *itens léxicos* e *combinações* desses para formar *sentenças válidas* em uma linguagem.
  - Uma gramática formal usa uma notação formal, isto é restrita e específica.
- Classes de gramáticas úteis para a teoria de compilação:
  - BNF (gramática livre de contexto),
  - Gramática regular.
- Uma linguagem é qualquer conjunto de strings (de tamanho finito) de caracteres escolhidos de um alfabeto fixo de símbolos finitos.

Modelos Formais para Tradução

Gramática Backus-Naur Form (BNF)

- Uma gramática BNF é composta de um conjunto *finito* de regras gramaticais BNF, descrevendo uma linguagem
  - É uma *metalinguagem* ou seja uma linguagem usada para descrever outras linguagens.
  - Gramática BNF é a definição formal da sintaxe de uma LP usando a notação BNF: *terminais*, ´::=´, ´<´, ´>´ ou ´|´.

::=	é definido como
<*ca*>	*ca* é uma categoria sintática, símbolo não terminal, definido em termos de outras entidades).
terminais	são palavras primitivas (itens léxicos) da LP.
|	ou

Gramática BNF

Notações equivalentes

- EBNF

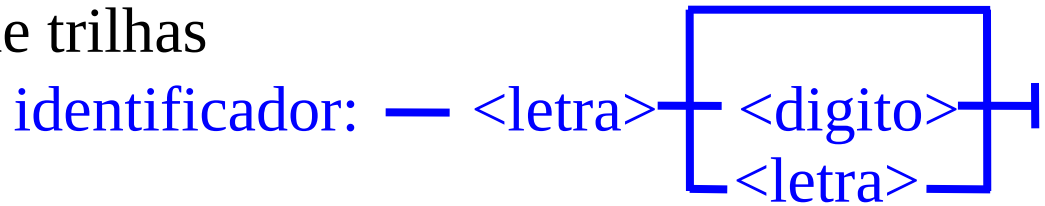
  [...] indica elementos opcionais.
  [|] indica alternativas de escolha.
  {...}* indica seqüência arbitrária de instâncias (*zero* ou *mais* repetições) de elementos sintáticos.

  <identificador> ::= <letra>{<letra>|<dígito>}*
  <inteiro> ::= [+|-]<dígito>{<dígito>}*

  Alguns extensões usam a notação {...}+ significando uma ou mais ocorrências do elemento sintático envolvido:

  <inteiro> ::= [+|-]{<dígito>}+

- Diagrama de trilhas



Gramática BNF

Notações equivalentes

- Regra de produção
  Obtida pela substituição de ::= pelo símbolo → e representação de não terminais como uma única letra maiúscula
  Exemplo:
  <X> ::= <B> | <C>
  X → B | C
- Dada uma gramática, pode-se utilizar uma regra de substituição simples para gerar strings válidas na linguagem:
  - substitua qualquer não terminal pela expressão no lado direito de uma regra de produção que o contenha no lado esquerdo.
    S → SS | (S) | () gera seqüências de parênteses corretas.
    P.ex.: S ⇒ (S) ⇒ (SS) ⇒ (()S) ⇒ (()())

Gramática BNF

Árvore de análise (parse)

- Uma string representa um programa sintaticamente válido em uma gramática BNF se passar, sem erro, por uma análise sintática, utilizando as regras da gramática. Essa análise gera uma árvore de parse.

  Exemplo:

  x=y+5*(z+x); e as regras

  < atribuição> ::= <variável> = <ea>

  <ea> ::= <term> | <ea> + <term> | <ea> - <term>

  <term>::=<primário> | <term>*<primário> | <term>/<primário>

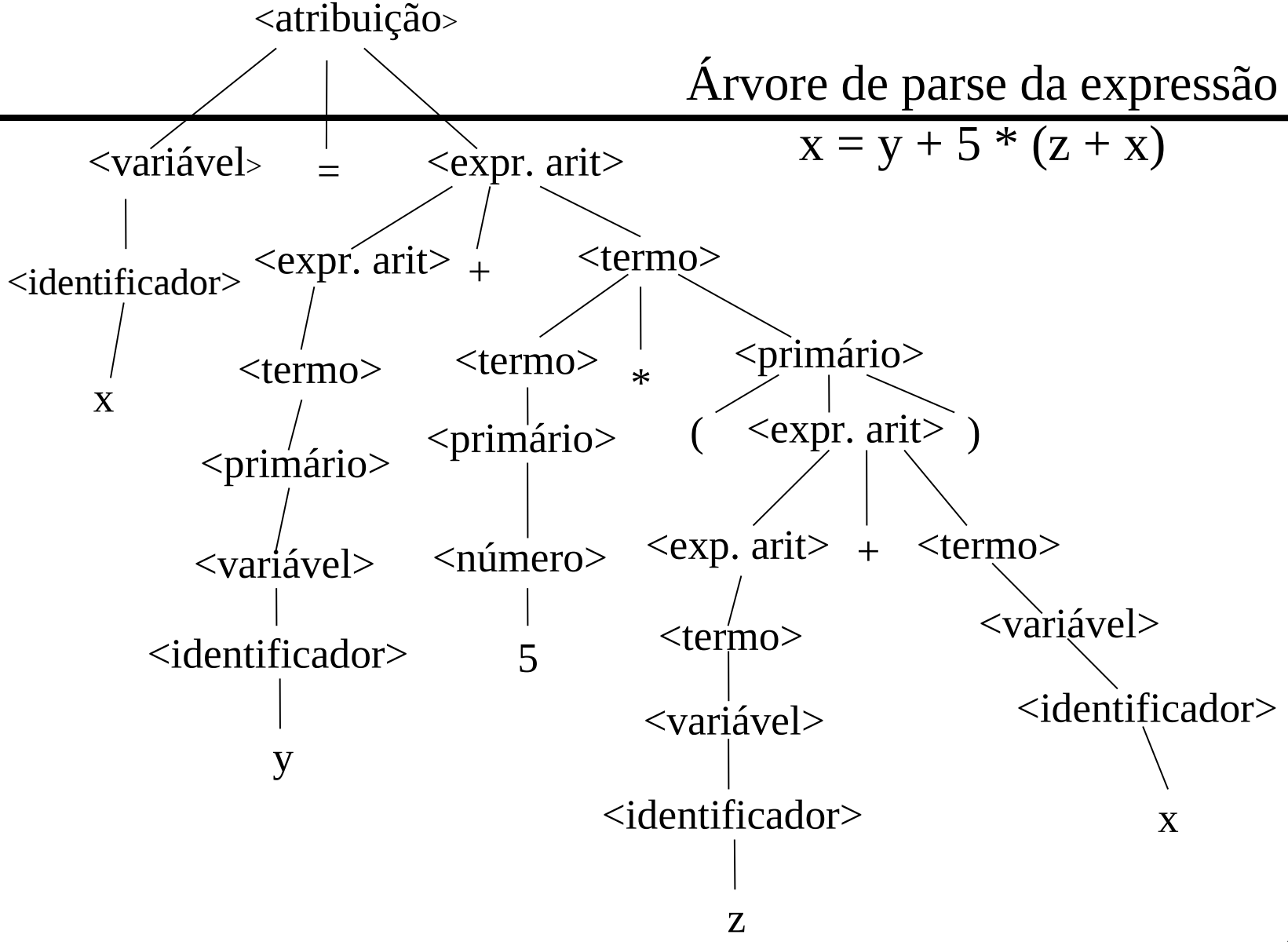
  <primário> ::= <variável> | <número> | (<ea>)

  <variável> ::= <identificador>[(<lista de subscritos>)]

  <lista de subscritos> ::= <ea> | <lista de subscritos>,<ea>

Árvore de parse da expressão

x = y + 5 * (z + x)



Gramática BNF

Limitações sintáticas

- A estrutura da BNF é simples e muito poderosa mas não consegue expressar regras sintáticas com *dependência contextual* do tipo:
  - o mesmo identificador não pode ser declarado mais de uma vez no mesmo bloco,
  - todo identificador deve ser declarado em um bloco envolvendo o ponto onde é usado,
  - um array deve ser referenciado com o mesmo número de subscritos com o qual é definido.
- BNF é uma gramática livre de contexto, isto é o lado esquerdo de uma regra só pode conter um símbolo.

Modelos Formais para Tradução

Autômato de estados finitos

- **Na fase de análise léxica o programa fonte é transformado numa seqüência de tokens (itens léxicos).**
- **Tokens podem ser reconhecidos por um modelo de máquina de estados também chamada autômato de estados finitos.**
- **Formalmente um FSA = (E, e_i, E_f, A_e, A_O), onde:**

  E = conjunto finito de estados (nós no grafo),

  e_i = estado inicial (um nó no grafo),

  E_f = conjunto de estados finais, $E_f \subset E$,

  A_e = alfabeto de entrada (rótulos para os arcos),

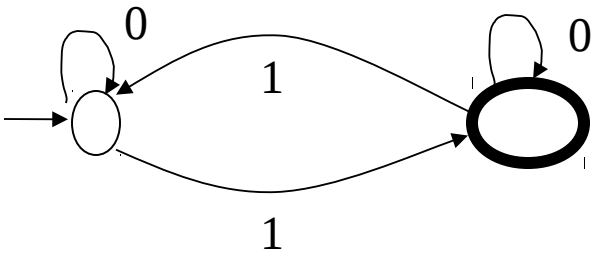
  A_O = conjunto de arcos orientados ligando elementos de E.

Cada nó pode ter *zero* ou mais arcos de *saída*, incluindo múltiplos arcos com o *mesmo* rótulo. Cada nó pode ter *zero* ou mais arcos de *chegada*.

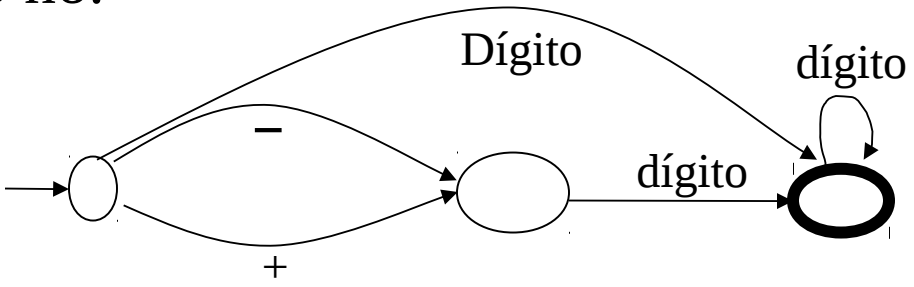
Autômato de Estados Finitos

Determinístico e não determinístico

- Determinístico: FSA *sem* repetição de arcos de *saídas* com o *mesmo rótulo* em um mesmo nó.

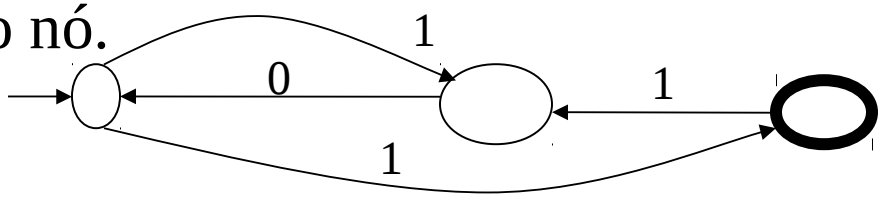


Reconhecedor de número com número impar de 1s.



Reconhecedor de: [+|-] <dígito>{<dígito>}*

- Não determinístico: FSA *com* repetição de arcos de *saída* com o *mesmo rótulo* em um mesmo nó.



Uma string é aceita pelo FSA não determinístico se existe **algum** caminho do nó inicial para o nó final, usando a string como entrada.

Autômato de Estados Finitos

Gramáticas regulares

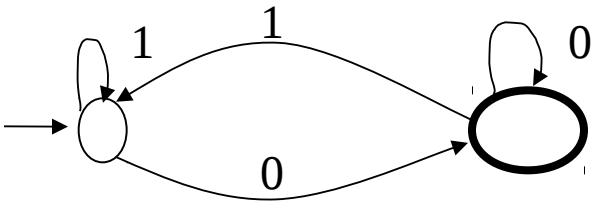
- Possuem regras da forma:
  <não-terminal> ::= terminal <não-terminal> | terminal
- São casos especiais de gramática BNF.
- O conjunto de linguagens *aceitas por FSA* é equivalente às linguagens *geradas por gramáticas regulares*.

Exemplo (geração de binários pares): <B> ::= 0<B>| 1<B> | 0

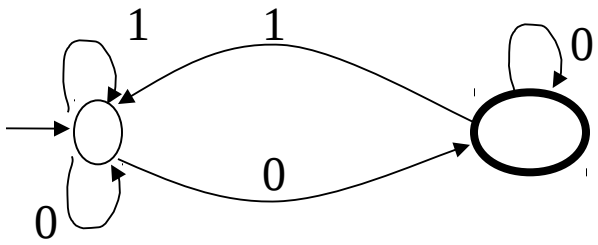
a) a regra de produção B → 0B| 1B | 0 gera binários pares

0B e 1B geram um binário qualquer. A produção 0 multiplica-o por 2.

b) FSA determinístico (gera/reconhece pares)



Determinístico



?

Autômato de Estados Finitos

Expressões regulares

- Terceira forma de definição de linguagens (equivalente a FSA e a gramática regular).
- <er> ::= terminal | <a> ∨ <b> | <a><b> | (<a>) | {<a>}*

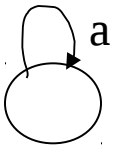
onde:

- um símbolo terminal é uma expressão regular,
- a∨b é a alternação das expressões regulares a ou b,
- ab é a concatenação das expressões regulares a e b,
- a*, fecho de Kleene, representa zero ou mais repetições de a (i.e. ε, a, aa, aaa, ...).

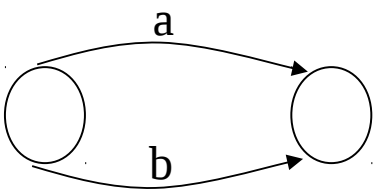
Autômato de Estados Finitos

Expressões regulares

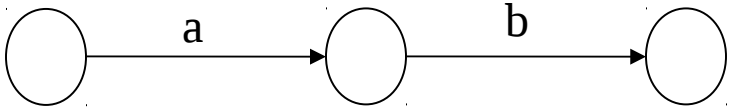
- Converter um FSA para uma expressão regular nem sempre é óbvio.
- Converter expressões regulares em um FSA é direto, com a aplicação das estruturas:



Fecho de Kleene: a*



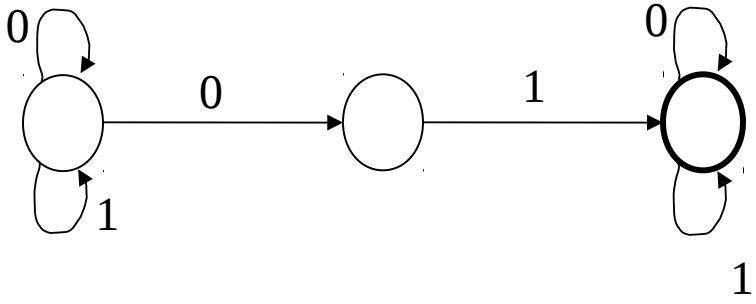
Alternação: a ∨ b



Concatenação: a b

Exemplo:

$(0 \vee 1)^{*}01(0 \vee 1)^{*}$



Modelos Formais para Tradução

Autômatos pushdown (PDA)

- Gerar strings em uma linguagem⇒ *gramáticas BNF*.
- Reconhecer strings na linguagem⇒ *autômato pushdown*
- PDA é um FSA com uma pilha associada.
- Os movimentos do PDA são os seguintes:
  1. um símbolo de entrada é lido e o topo do stack também é lido;
  2. os dois símbolos são comparados. O PDA entra em um novo estado e escreve zero ou mais símbolos na pilha;
  3. a aceitação de uma string ocorre se a pilha ficar vazia (ou de forma equivalente, se o PDA atingir um estado final).
- Para reconhecer $a^n b^n$, empilhe a. Para cada entrada b, desempilhe um a. A string é aceita se o término dos b ocorrer com pilha vazia.

Autômatos Pushdown

PDA não determinístico

- PDA não determinístico é um autômato que possui mais de um estado com a mesma entrada.
  Uma string é aceita se existe uma seqüência possível de movimentos que aceita a string.
- PDA determinístico e não determinístico são DIFERENTES

Palindromes S → 0S0 | 1S1 | 2 são reconhecidas pelo PDA determinístico, com os seguintes movimentos:

1. empilhe todos os 1 e 0 lidos.
2. mude de estado com 2.
3. desempilhe se cada entrada for igual ao topo da pilha.
4. a string é aceita se encerrar a entrada com stack vazia.

Autômatos Pushdown

PDA não determinístico

- Exemplo:
  S → 0S0 | 1S1 | 0 | 1 gera 011010110?
  S⇒0S0 ⇒ 0 1S1 0 ⇒ 01 1S1 10 ⇒ 011 0S0 110 ⇒ 0110 1 0110
  PDA não determinístico reconhece essa string? Somente se apostar (chutar) onde está o meio!

stack	meio?	compara stack com	
ε	0	11010110	
0	1	1010110	
01	1	010110	
011	0	10110	
0110	1	0110	correto!

- PDA não determinístico reconhece qualquer gramática livre de contexto.

Modelos Formais para Tradução

Algoritmos eficientes de parsing

- Uma gramática descreve a estrutura do programa de um modo top-down especificando todo o programa, depois os sub-programas, comandos, declarações, etc.
- O parser atua no sentido bottom-up para gerar uma árvore de parse a partir de uma seqüência de tokens, representando o fonte:

  a estrutura é construída analisando a entrada (tokens) da esquerda para a direita (LR), à medida que os tokens são lidos.

  a seqüência de tokens pode não constituir um programa válido!
- Cada tipo de gramática formal está relacionado com um tipo de autômato.

Algoritmos Eficientes de Parsing

Estratégia Geral

- Um *autômato* é uma máquina abstrata capaz de ler uma fita com uma seqüência de caracteres e produzir uma fita de saída com outra seqüência de caracteres.
- Se a BNF for ambígua o *autômato* é *não determinístico*:
  existem opções de movimento e o autômato deverá apostar qual é o mais apropriado em um certo instante.
- Tradução de linguagem requer autômato determinístico.
- Gramática regular: sempre existe um autômato determinístico equivalente.
- Gramática BNF não ambígua: foi desenvolvido o *parsing recursivo descendente* para reconhecer sentenças válidas.

Algoritmos Eficientes de Parsing

Parsing recursivo descendente

- Gramática BNF não ambígua, reconhecida por PDA determinístico, pode ser descrita por gramática LR (left to right), desenvolvida pelo Knuth.
  - SLR (simple LR) e LALR (look ahead LR): subclasses de gramáticas LR com algoritmos de parsing eficientes.
- LP atuais usam gramáticas SLR, LR ou LL para que os parsers possam ser obtidos automaticamente via YACC.
- LR(k): gramáticas que olham k símbolos a frente, da esquerda para a direita, para tomar decisão de parsing.
  - LR(1): gramáticas que precisam olhar apenas um símbolo a frente para decidir qual o elemento sintático encontrado.

Parsing recursivo descendente

- Parsing é o processo de construir uma árvore de análise para um dado string de entrada:
  - em geral não analisam tokens.
  - um parser recursivo descendente constrói uma árvore de parse usando uma abordagem top-down.
  - cada não terminal na gramática tem um subprograma associado a ele que analisa as formas sentenciais que o não terminal pode gerar.
  - os subprogramas de parsing recursivo descendente não podem ser construídos a partir de gramáticas recursivas à esquerda.

Parsing recursivo descendente

Exemplo: expressão aritmética

- BNF ↔ linguagem infinita

<ea> ::= <term> | <ea> + <term> |
<ea> - <term>

- EBNF ↔ representação equivalente mais adequada para parsing.

<ea> ::= <term>{[+|-]<term>}*

<term>::=<primary>{[*|/]<primary>}*

- Se reconhece um termo. Caso seja seguido de + ou -, se reconhece um novo termo e assim sucessivamente.
- Convenções
  - nextchar: 1º caracter do não terminal.
  - getchar: ler um caracter da entrada.
  - Identifier: scanner p/ ler identificador.
  - Number: scanner p/ ler número.

```
procedure AssingStmt
  begin
  Variable;
  if nextchar <> '=' then error
     else begin nextchar := getchar;
     Expression  end
  end;
procedure Expression
  begin
     Term;
     while ((nextchar='+') or (nextchar='-')
     do begin nextchar:=getchar;
     Term end
end;
```

Parsing recursivo descendente

Exemplo: expressão aritmética

```
procedure Term;
begin
    Primary;
     while (nextchar =‘*’) or (nextchar=‘/’) do
       begin  nextchar := getchar; Primary end
end;
procedure Variable;
begin
    Identifier;
    if nextchar = ‘(‘ then begin
      nextchar := getchar;  SubsList;
      if nextchar = ‘)’ then nextchar := getchar
      else Error  /*  falta ‘)’  */
    end
end;
```

```
procedure Primary;
begin
    if nextchar=letter then Variable
    else if nextchar=digit then Number
    else if nextchar = ‘(‘ then
          begin
          nextchar := getchar;
           Expression;
          if netxchar = ‘)’ then nextchar
                                  := getchar
          else Error     /* falta ‘)’ */
          end
     else Error  /* falta ‘(‘ */
end;
```

Modelos Formais para a Tradução

Modelagem Semântica

- O manual de uma LP precisa definir o significado de cada construção, isolada e em conjunto com outras.
  - Sintaxe: gramática formal
  - Semântica: linguagem natural e exemplos, o que dá margem a mal entendidos devido a ambigüidades de significado.
  - $\Rightarrow$ Método: definição precisa, concisa e legível da semântica.
- O problema da semântica tem sido objeto de estudos, mas um método satisfatório para definir a semântica de uma LP ainda não foi encontrado.

Modelagem Semântica

Exemplos de métodos propostos

- Modelos gramaticais.
  - Agregam extensões (semânticas) a uma gramática formal.
- Modelos imperativos ou operacionais.
  - Descrevem a semântica através de autômato complexo representando o computador virtual onde a LP é executada.
- Modelos aplicativos.
  - Abordagem do paradigma funcional para descrição semântica.
- Modelos axiomáticos.
  - Estende o cálculo de predicados para incluir programas.
- Modelos de especificação.
  - Relacionamento entre as funções que constituem o programa.

Modelagem Semântica

Modelos Gramaticais

- Uma das primeiras tentativas de criação de modelos formais para a semântica.
- Consiste em adicionar extensões às BNF que definem a LP, baseado na extração de informações semânticas da árvore de parse.
- *Gramática de atributos* (Knuth, 68) é uma forma de extrair informações adicionais a partir da árvore de parse.
  - A idéia é associar uma função (atributo) a cada nó da árvore de parse do programa, representando o conteúdo semântico do nó.

Modelos Gramaticais

Gramática de Atributos

- Semântica estática
  - regras se relacionam apenas indiretamente com o significado dos programas durante a execução.
  - durante a compilação se pode verificar se tais regras foram atendidas ou não.
  - elas relacionam com as formas legais dos programas mas não podem ser descritas por uma BNF.
    - regras de tipificação (checagem de tipos)
    - variáveis devem ser declaradas antes de serem referenciadas
    - identificador seguindo o end de procedimento Ada deve coincidir com o nome do procedimento.

Modelos Gramaticais

Gramática de Atributos

- A gramática de atributos é criada pela adição de funções (*atributos*) a cada regra na gramática BNF.

  $X_0 \rightarrow X_1 \ldots X_n$

- *Herança de atributo*: atributo herdado é uma função que relaciona valores de não-terminais na árvore com valores de não-terminais representados em nós (ancestrais) mais próximos da raiz:

  $H(X_j) = f(A(X_0), \ldots, A(X_{j-1}))\ ,\ j = 1, .., n$

- *Atributo sintetizado* é uma função que relaciona não-terminais na esquerda com valores de não-terminais na direita em uma regra. Os atributos são sintetizados a partir das informações mais distantes da raiz (filhos) e passados na direção da raiz.

  $S(X_0) = f(A(X_1), \ldots, A(X_n))$

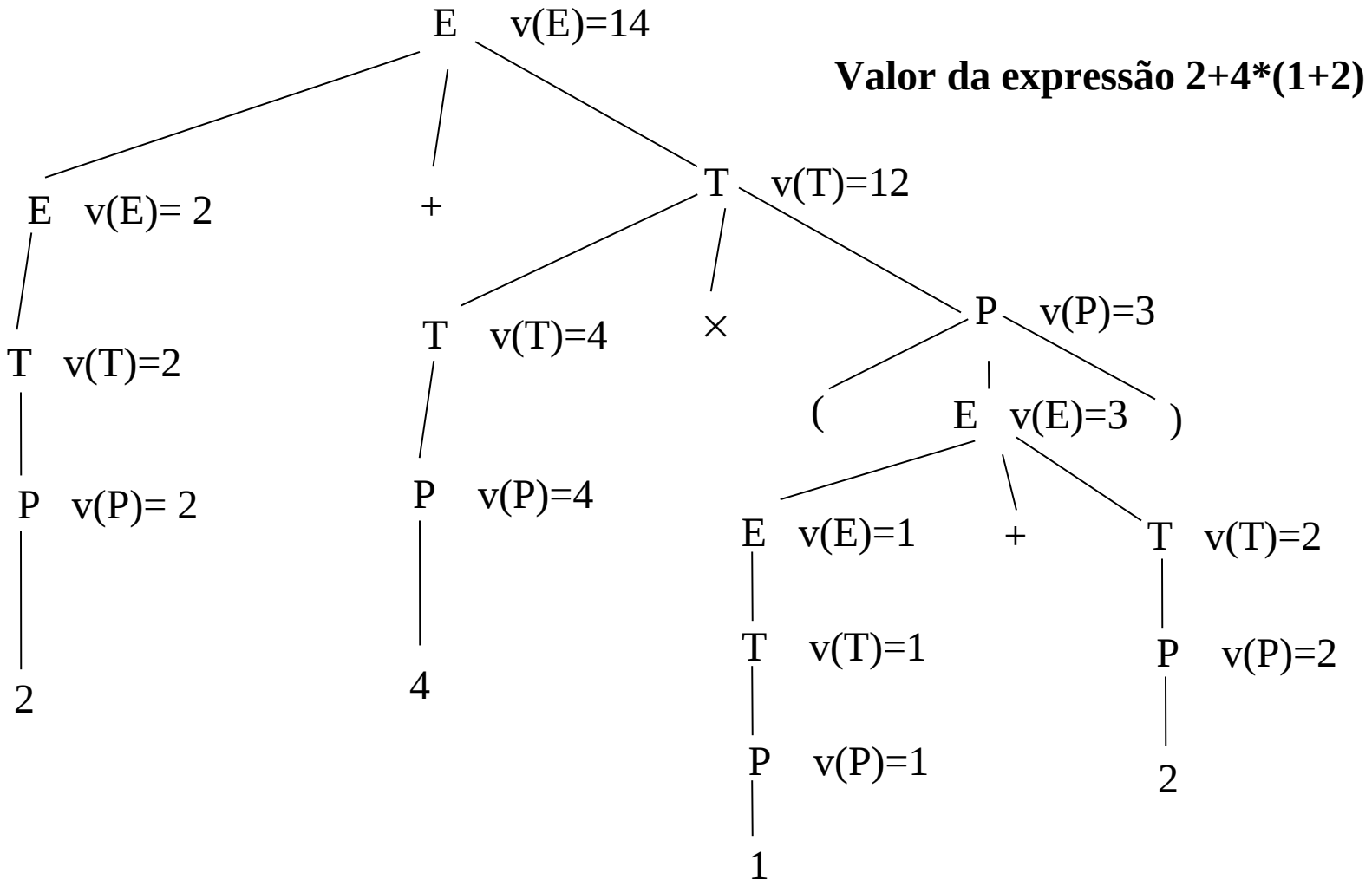
Exemplo de Gramática de Atributos

- **Gramática simples para expressão aritmética:**
  $E \rightarrow T|E+T$ $T \rightarrow P|T\times P$ $P \rightarrow I|(E)$
- **A semântica dessa LP é definida pelas seguintes funções que produzem o valor de qualquer expressão gerada por essa gramática:**

Produção	Atributo sintetizado
$E \rightarrow T$	valor(E) = valor(T)
$E \rightarrow E+T$	$valor(E_1) = valor(E_2) + valor(T)$
$T \rightarrow P$	valor(T) = valor(P)
$T \rightarrow T\times P$	$valor(T_1) = valor(T_2) * valor(P)$
$P \rightarrow I$	valor(P) = valor do número I
$P \rightarrow (E)$	valor(P) = valor(E)

Gramática de Atributos

Exemplo de árvore com atributos



Gramática de Atributos

- **Gramática de atributos pode ser utilizada para passar informação semântica para estruturas sintáticas, facilitando o processo de analise e geração de código.**
  - **Informação de declaração pode ser coletada em produções de declarações e passadas para a geração de código em e.a.**

Produção	Atributo sintetizado
<dcl> ::= <decl><dcl>	$decl\_set(dcl_1) = decl\_id(decl) \cup decl\_set(dcl_2)$
<dcl> ::= <decl>	decl_set(dcl) = decl_id(decl)
<decl> ::= declare x	decl_id(decl) = x
<decl> ::= declare y	decl_id(decl) = y
<decl> ::= declare z	decl_id(decl) = z

- **Se tem apenas atributos sintetizados, eles podem ser avaliados durante a geração da árvore de parse. Este é o caso do YetAnotherCompilerCompiler.**

Modelagem Semântica

Modelo imperativo (ou operacional)

- Computador virtual é descrito como um autômato complexo com estados correspondendo aos estados do programa em tempo de execução (valores das variáveis, código executável e estruturas de housekeeping definidas pelo sistema).
  - Operações que alteram os estados do autômato são descritas de forma formal (correspondem a execução de instruções).
  - Uma parte da definição, descreve como o programa é traduzido para um estado inicial do autômato.
  - A partir desse estado inicial, o autômato se move de estado para estado, segundo suas operações, até chegar ao estado final.
- VDL tem enfoque operacional. Ela estende a árvore de parse para incorporar um interpretador. Estado = árvore de pgm + árvore de dados na máquina. Cada instrução move de um estado para outro.

Modelagem Semântica

Modelo Aplicativo

- Tenta construir (hierarquicamente) a definição da função que o programa, codificado em uma certa LP, executa.
- Cada operação, primitiva ou definida pelo usuário, representa uma função matemática.
- As estruturas de *controle de seqüência* são usadas para compor essas funções em seqüências maiores, representadas no texto do programa por comandos e expressões.
- Loops são tratados como funções recursivas, construídas a partir dos componentes do corpo do loop.
- Por último, é derivado um modelo funcional do programa inteiro.
- O método da *Semântica Denotacional* de Scott e Strachey e o da *Semântica Funcional* de Mills são exemplos desse enfoque.

Modelagem Semântica

Modelo Axiomático

- Define a semântica de cada construção sintática na LP como um *axioma* ou *regra de inferência* do cálculo de predicados:
  - usada para deduzir o efeito de execução da construção sintática.
- Para entender o programa inteiro, usa-se os axiomas e regras de inferências como em provas matemáticas.
  - Supondo que as variáveis de entrada obedecem a certas restrições, os axiomas e regras de inferências são usados para achar restrições atendidas pelos valores de outras variáveis após a execução de cada comando no programa.
  - Com esse procedimento pode-se eventualmente prova que os valores de saída representam funções apropriadas calculadas a partir dos valores de entrada.
- Um exemplo desse método é a *Semântica Axiomática* de Hoare.

Modelagem Semântica

Modelos de Especificação

- Descreve o relacionamento entre as várias funções que implementam o programa.
- A implementação é correta (do ponto de vista da especificação) se quaisquer duas funções do programa, obedecem o relacionamento estabelecido para elas.
- Tipos algébricos de dados são uma implementação formal.
  - Na construção de um programa que implementa a pilha P, push e pop são operações inversas.
  - Um axioma que pode ser estabelecido é:
    pop(push(P,x)) = P, e x por efeito colateral.
  - Qualquer implementação de pilha que preserve esta propriedade (entre outras) é uma implementação correta.

Modelagem Semântica

Conclusão

- Definição semântica formal: parte da especificação de uma LP?
  - PL/I: inclui especificação VDL da semântica dos comandos
  - Ada: inclui semântica denotacional
- Nenhum modelo semântico provou ser útil para implementadores e usuários simultaneamente.
  - Modelo *imperativo (operacional)*: descrição da VM é útil para o implementador mas muito detalhada e sem valor para o usuário.
  - Modelo *aplicativo (funcional, denotacional)*: são complexos e sem muita utilidade para implementador e usuário.
  - Modelo *axiomático*: pode ser entendido pelo usuário, mas a complexidade aumenta se aplicada a definição de toda LP inteira. Sem utilidade para o implementador.