V.2 – Especificação Sintática de Linguagens de Programação

- **Deve ser baseada:**
  - **No planejamento da Linguagem / Compilador**
    - **Objetivos, Filosofia, Potencialidades, ...**
  - **Nos critérios de projeto/avaliação**
    - **Legibilidade, Confiabilidade, etc...**

- **Deve ser efetuada em duas etapas:**
  1. **Especificação Preliminar**
     - **Informal**
     - **Construções/Recursos necessários**
     - **Mecanismos de Abstração (dados/funcional)**
     - **Declarações, Comandos e Expressões**
     - **Potencialidades desejadas**

2. Especificação Detalhada

- Preferencialmente FORMAL
- Definição do Mecanismo de Especificação
  - GLC, PDA, ERE
- Definição da Notação
  - BNF (pura, estendida)
  - Diagramas Sintáticos
  - RRP (Produções com lado direito regular)
- Escolha do Mecanismo e da Notação
  - No caso de uso de Geradores de A.S.:
    - Mecanismo / Notação próprio
  - No caso de Implementação Manual
    - Livre escolha, podendo ser influenciada pela Técnica de Análise usada.

Vantagens da Especificação Formal da Sintaxe de Linguagens de Programação

- Precisão
- Legibilidade
- Algoritmos de análise eficientes
- Implementação automática de PARSER's
- Verificação automática da adequação da especificação para a técnica escolhida
  - Auxílio na detecção de ambiguidades
    - A partir dos conflitos na tabela de parsing
  - Detecção de Recursão à esquerda e Não-fatoração
- Facilidade para teste da Consistência da especificação
  - presença de símbolos inúteis
  - construções reconhecidas X const. desejadas
    - uso de simuladores, casos de teste
- Facilidade para detecção/recuperação de erros
- Possibilidade de implementação de esquemas de tradução dirigidos pela sintaxe
- Favorece a extensibilidade de Linguagens
- Favorece a corretude do compilador como um todo

V.3 – Implementação de PARSER's

- Estrutura do Analisador
  - Depende da estrutura do compilador
    - Núcleo do compilador
      - comanda demais procedimentos
    - Procedimento independente
      - mesmo nível das demais fases
      - necessidade de módulo integrador
- Estratégia de implementação
  - Uso de Geradores de PARSER's
    - Qual? (ex. YACC, GAS, GALS ...)
    - Oque utilizar?
      - O analisador todo ou só as tabelas?
  - Programação manual
    - Baseada em especificação formal?
    - Usando técnicas formais?
      - forma de obtenção da tabela de parsing
      - Descendente Recursivo - + comum!

- **Escolha da técnica de análise usada**
  - **Necessidade de adequação da especificação (da GLC) às exigências da técnica**

    Exemplos:
    - **Para qualquer técnica determinística:**
      - **A GLC usada não pode ser ambígua**
    - **Para técnicas descendentes:**
      - **A GLC não pode possuir Recursão a Esquerda e tem que estar Fatorada.**

  - **Integração com o Analisador Semântico e com o Gerador de Código**
    - **Uso de ações semânticas embutidas na GLC**
    - **Ações dependem da técnica utilizada**

- **Uso de Geradores de PARSER's**
  - **Automatizam a geração de analisadores sintáticos**
  - **Baseados na especificação formal da Linguagem**
  - **Facilitam a validação do compilador**

Exemplos de Geradores de PARSER

YACC – Yet Another Compiler Compiler
(S. C. Johnson, 1975)

- Implementa a técnica LALR(1)
- Uso integrado YACC / LEX
- Semântica incluída através de comandos (C)
- Permite a solução de conflitos decorrentes da ambiguidade da GLC
- Universalmente utilizado

Estrutura do YACC

Declarações
%%
Regras de tradução
%%
Rotinas auxiliares

Exemplo de uma especificação YACC
(calculadora de mesa simples)

G: E → E + T | T
T → T * F | F
F → (E) | digito

```
% { #include <ctype.H> } %
%  token digito

%%
line : E '\n' {printf ($1)}
E    : E  '+'  T      { $$ = $1 + $3;}
     |  T             { $$ = $1 }
     ;
T    : T  '*'  F      { $$ = $1 * $3;}
     |  F             { $$ = $1;}
     ;
F    : '('  E  ')'    { $$ = $2;}
     |  digito        { $$ = $1;}
     ;

%%

YYLEX ( )  (* analisador léxico *)
{  int c;
   c = getchar(c);
   if (isdigit(c))
      { YYVAL = c – '0';
        return digito }
   return c; }
```

GAS – Gerador de Analisadores Sintáticos

(trab. de graduação - Edson Linhares, CCO/UFSC – 91)

- Engloba aspectos conceituais – objetivos didáticos
- Diversas técnicas – (Preditiva (LL(1)), LR(1), SLR(1) e LALR(1))
- Várias linguagens (Pascal, C e Modula2)
- Simulador integrado
- Fornece documentação (gramática, first e follow, tabelas, conflitos)
- Possibilita resolução de conflitos (de forma interativa)
- Integração com A. Semântico e G. Código
  - através da introdução de AÇÕES SEMÂNTICAS (verificação e tradução) na GLC
- Interface pobre (DOS)
- Nova versão (for Windows) – BRAIN
  - trabalho de conclusão do CCO/UFSC
  - integrada com um G.A. L. (Micro-LEX)
  - gera analisadores em Delphi, C++ e JAVA

Exemplo de uma especificação GAS

(calculadora de mesa simples)

G: E → E + T | T
T → T * F | F
F → (E) | digito

gramática expressao;
Vn = { <E'> , <E> , <T> , <F> }
Vt = { '+' , '-' , '(' , ')' , digito }
S = <E'>
P = { <E'> ::= <E> #7 '$' ;
<E> ::= <E> '+' <T> #6
| <T> #5 ;
<T> ::= <T> '*' <F> #4
| <F> #3 ;
<F> ::= '(' <E> #2 ')'
| digito #1 ; }

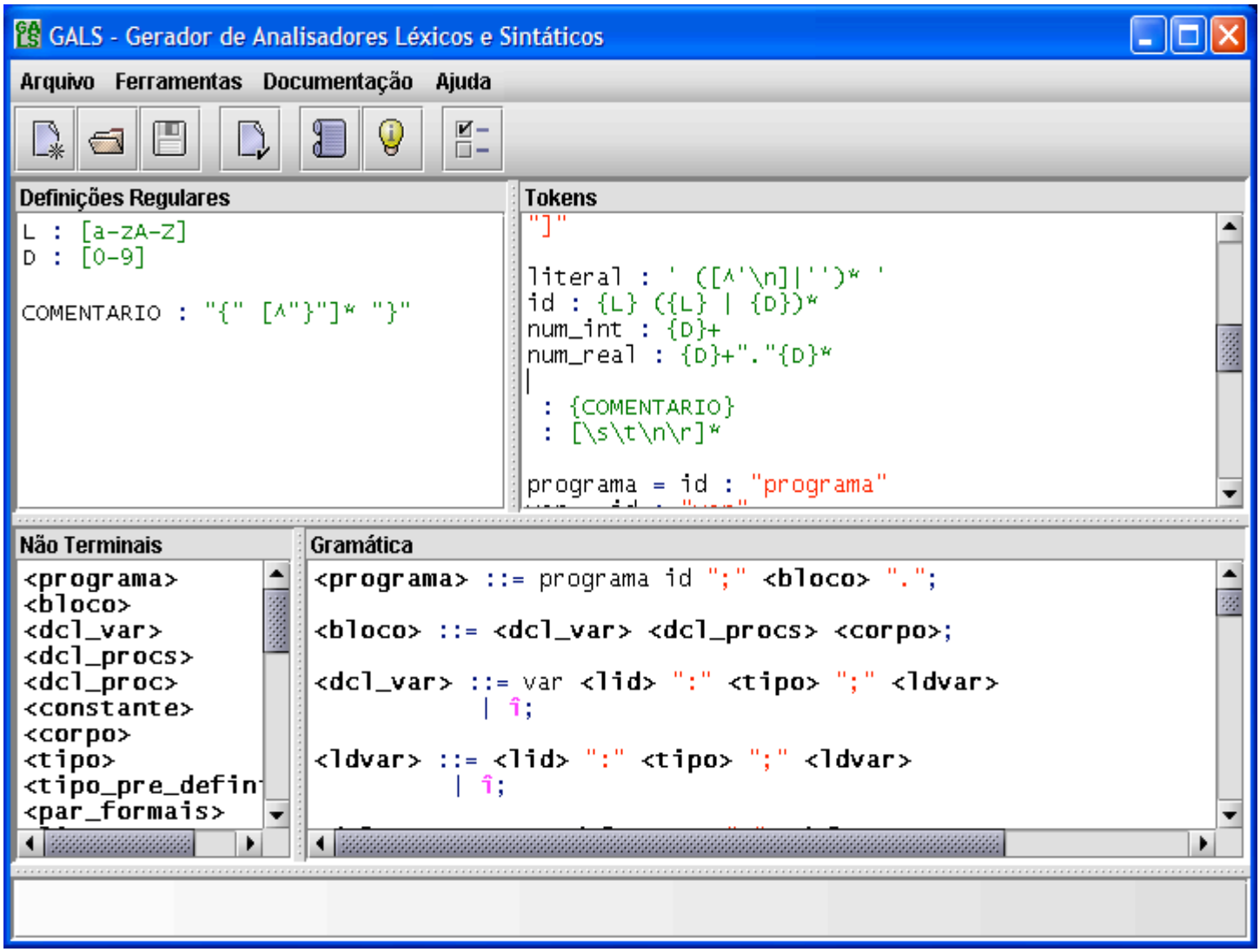
Desta forma, o semântico (interpretador), seria algo como:

Ação #1 : F.val := valor do digito
Ação #2 : F.val := E.val
Ação #3 : T.val := F.val
Ação #4 : T.val := T.val * F.val
Ação #5 : E.val := T.val
Ação #6 : E.val := E.val + T.val
Ação #7 : write('valor da expressao = ', E.val}

OBS.: F, T, e E são variáveis do tipo registro, contendo o campo "val", declaradas explicitamente no analisador semântico.

GALS

(trabalho de Graduação – CCO/UFSC – Carlos E. Gesser)



V.4 - Especificação Sintática da LSI-132

Considerações preliminares

Objetivo didático

Paradigma estruturado

Estilo PASCAL, simplificado
(apenas construções básicas, nativas)

Especificação preliminar

- **Identificação das construções**

  Declarações

  Constantes e variáveis

  Procedimentos (subprogramas)

  Comandos

  Atribuição, E/S, seleção e repetição

  Expressões

  operandos e operadores convencionais

  Recursos avançados

  nenhum (apenas recursos básicos da Ling.)

- **Detalhamento (informal) das construções**

  Declarações:

  Variáveis

  - declaração explicita
  - tipos pré-definidos
    - inteiro, real, booleano e caracter
  - tipos definidos pelo usuário
    - cadeia (string) de caracteres
    - vetor (array) uni e bi-dimensional, com elementos de tipo pré-definido

Constantes

- **Tipos pré-definidos**

Procedimentos

- **funções e procedures**
  - **com e sem parâmetros**
  - **parâmetros**
    - **por valor e por referência**
    - **apenas de tipos pré-definidos**
  - **permitir procedimentos aninhados**
  - **permitir declarações locais de tudo**
  - **funções apenas de tipos pré-definidos**

Comandos

Atribuição

Chamada de procedimento

E/S: (sem arquivos e formatação)

- **Entrada – lista de identificadores**
- **Saída – lista de variáveis, constantes e expressões**

Seleção: se-então, (if-then)

se-então-senão (if-then-else)

Repetição: enquanto-faça (while-do)

Comando composto: { "lista de comandos" }

Comando vazio: ε

Expressões:

Operadores

aritméticos:

Binários: +, -, *, /

Unários: -

lógicos: e, ou, não

relacionais: =, <, >, <=, < >, < >

Operandos

variáveis: simples ou indexadas

constantes: numéricas (inteiras e reais), booleanas (verdadeiro/falso), literais (cadeia ou char)

designadores de função

Especificação formal

- **Mecanismo: GLC**
- **Notação: BNF pura (a ser usada no GALS)**

V.5 – Implementação do PARSER da LSI-132

- **Técnica de análise sintática: PREDITIVA (LL(1))**
- **Estrutura: Núcleo do compilador**
- **Estratégia: Uso do gerador GALS**
- **Adaptar o Léxico (de acordo com a GLC padrão)**
- **Integrar Léxico e Sintático (via GALS)**
- **Linguagem de implementação: mesma do léxico**
- **Equipes: 2 componentes** (mesma equipe que implementou o léxico)

Trabalho complementar

- **Criar uma nova linguagem que estenda a LSI-132 com:**
  - **Um novo tipo pré-definido**
  - **Um novo tipo estruturado**
  - **Um novo comando**
  - **Recursos de OO, transformando-a em uma Linguagem multi-paradigma.**

- **Validar a gramática estendida, segundo a técnica de análise sintática LL(1), usando o GALS.**

- **Entregar arquivo .gals para avaliação e um programa exemplo contento todas as extensões propostas.**