UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA DE CAMPINAS
DEPARTAMENTO DE AUTOMAÇÃO (FEE)

UM SISTEMA GERADOR DE COMPILADORES

Mario Jino

Orientador: Nelson C. Machado

Tese de Mestrado apresentada à Faculdade de Engenharia de Campinas da Universidade Estadual de Campinas

Janeiro - 1974

A G R A D E C I M E N T O S

Desejo agradecer ao Prof. Dr. Nelson C. Machado pelo auxílio durante a elaboração deste trabalho e pelas sugestões na preparação deste documento.

Desejo também agradecer a todos que, direta ou indiretamente, contribuiram para a realização desta tese.

R E S U M O

Neste trabalho é apresentada a descrição de um sistema gerador de compiladores que, a partir da especificação sintática de uma linguagem, produz tabelas que dirigirão a análise sintática de programas da linguagem. A análise sintática é efetuada por dois analisadores fixos: um analisador "top down" de descida recursiva para a parte geral da linguagem e um analisador "bottom up" de precedência de operadores para sub-conjuntos específicos da linguagem.

São descritas a especificação sintática das linguagens aceitas pelo sistema, a construção das tabelas e os analisadores . A especificação semântica deve ser feita pelo próprio usuário do sistema.

I N D I C E

1. INTRODUÇÃO

Construção de compiladores tem grande importância prática e teórica na área de computação digital, demandando muito trabalho e tempo de quem se dedica a esta tarefa. Grande parte do esforço envolvido no projeto de sistemas de computação é gasto no desenvolvimento de compiladores. Principalmente durante a década passada, foram desenvolvidos vários esquemas para facilitar o trabalho de quem escreve compiladores, através da automação de diferentes partes deste trabalho. Esses esquemas, implementados como um programa ou um conjunto de programas, são conhecidos como sistemas geradores de compiladores.

Um gerador de compiladores é basicamente um sistema que, dadas as descrições sintática e semântica de uma linguagem de programação, pode produzir um compilador para esta linguagem.

Define-se sintaxe de uma linguagem como uma especificação dos comandos bem formados dessa linguagem e, semântica como uma especificação de como estes comandos devem ser executados por um computador. Portanto, sintaxe refere-se à forma e semântica ao significado dos comandos.

Um gerador de compiladores completo seria composto de:

1) Um processador sintático que, dada a descrição formal de sintaxe de uma linguagem, produz um analisador sintático para a linguagem.

2) Um processador semântico que, dada a descrição formal da semântica de uma linguagem, produz um conjunto de rotinas capazes de gerar código para cada construção válida da linguagem e de construir as tabelas necessárias para a execução de um programa.

Um compilador seria, por conseguinte, constituido de um analisador sintático e de rotinas semânticas.

Neste trabalho visamos apenas a geração automática do analisador sintático, isto é, nosso objetivo é o processador sintático.

Na análise sintática, existem duas estratégias clássicas:

a) "Bottom up" (ascendente) ou determinística em que os símbolos de entrada são examinados um de cada vez, tentando combinar os símbolos já examinados com o novo símbolo para formar combinações mais complexas da linguagem. A análise é feita partindo-se dos símbolos do programa fonte e o conjunto dos símbolos encontrados determina alternativas possíveis para o próximo símbolo a ser encontrado. Se o novo símbolo não se enquadra em uma das alternativas possíveis a cadeia de símbolos de entrada não pertence à linguagem.

b) "Top down" (descendente) ou não determinística em que se faz uma análise por tentativas. São procuradas na entrada cadeias de símbolos que satisfaçam uma certa configuração estabelecida a priori. Se a busca não é bem sucedida estabelece-se uma outra configuração como objetivo de busca e tenta-se novamente. Na análise parte-se de objetivos gerais de busca (o primeiro objetivo de busca é um programa da linguagem) e, a cada passo da análise, são estabelecidos sub-objetivos de ordem inferior contidos nos objetivos. Quando um sub-objetivo de um objetivo não é encontrado tenta-se outra alternativa do objetivo. Se não houver mais alternativas a busca do objetivo corrente falhou e retorna-se ao objetivo de ordem superior que o contém, tentando-se outra alternativa deste. Está claro que o mecanis

mo de retorno a objetivos de ordem superior inicia-se sempre que falha a busca de um símbolo do programa fonte. Só se pode concluir que a cadeia de símbolos de entrada não pertence à linguagem quando falham todas as alternativas do primeiro objetivo (programa).

Os analisadores sintáticos "bottom up" são rápidos e contêm, no próprio algoritmo de análise, mecanismos de detecção de erros no programa fonte. Contudo, a notação formal mais comum utilizada na descrição da sintaxe de uma linguagem ("Backus-Naur Form" [1]) não pode ser utilizada diretamente na construção de analisadores "bottom up". É necessário utilizar métodos mais sofisticados para descrever a sintaxe: "tabelas de precedência de operadores" [2] , "matrizes de transição" [3] [4] , "linguagem de produção de Floyd" [5] , que são de utilização difícil e podendo mesmo dificultar a compreensão da linguagem.

Geradores "top down" são rápidos e podem utilizar notações formais simples e fáceis de serem entendidas (por exemplo, BNF com algumas modificações e restrições). Desvantagens surgem em relação à lentidão dos analisadores gerados e à dificuldade de inserção de mecanismos de detecção de erros. Os trabalhos mais antigos sobre geração de compiladores usam técnicas "top down" - Brooker & Morris [6] , Irons [7] .

O gerador de compiladores desta tese baseia-se na idéia contida no trabalho de Cheatham & Sattley [8] : análise "top down" para a parte geral da linguagem e análise "bottom up" para sub-conjuntos específicos da linguagem. A parte geral de uma linguagem (definições de comandos, declarações de tipos, etc.) tem uma estrutura sintática simples e pode ser analisada, sem muita desvantagem, por uma técnica "top down". Sub-conjuntos específicos de estrutura sintática mais complexa, como expressões aritméticas, são analisados com maior eficiência pelos mecanismos sofisticados de uma técnica "bottom up". A divisão da análise tem por objetivo aproveitar as melhores características de cada estratégia.

Foram estudados e comparados alguns métodos "top

down" quanto à simplicidade de implementação, velocidade de construção de compiladores e de análise sintática e espaço ocupado pelos analisadores: Trout [9], Earley [10], Cheatham & Sattley [8]. Decidimo-nos pela técnica de análise dirigida por tabelas [8], com uma implementação semelhante à de Gaffney [11]: um analisador fixo dirigido por tabelas geradas por um "CONSTRUTOR".

Visando a implementação dos sub-conjuntos "bottom up" foi feito um estudo semelhante compreendendo as seguintes técnicas: precedência de operadores de Floyd, linguagens de produção de Floyd, matrizes de transição, precedência simples e precedência generalizada, descritas em Feldman & Gries [12]. Precedência de operadores mostrou-se a melhor opção em vista da eficiência do método e aplicabilidade aos sub-conjuntos bem estruturados a que se destina. Para a construção das tabelas de precedência escolhemos o método de operações com matrizes booleanas (obtidas da descrição dos sub-conjuntos), Gries [13] [14].

2. VISÃO GLOBAL DO SISTEMA

Neste capítulo descrevemos resumidamente as funções principais do CONSTRUTOR e dos compiladores gerados.

Na Figura 1 mostramos o esquema geral do sistema gerador de compiladores tal como foi implementado. Foi implementado inicialmente apenas um sub-conjunto "bottom up", reservado no exemplo às expressões aritméticas.

2.1 O CONSTRUTOR

O CONSTRUTOR é essencialmente um algoritmo para a construção das tabelas que dirigirão a análise sintática da linguagem cuja descrição lhe foi fornecida.

Suas funções fundamentais são :

- Análise da descrição sintática geral e geração das tabelas de símbolos e da tabela de definições sintáticas. Isto é feito por um algoritmo contido no corpo do programa principal do CONSTRUTOR.

- Análise da descrição de sub-conjuntos "bottom up" e construção da tabela de símbolos terminais e da matriz de precedência de operadores. A sub-rotina CONSTRUTOPER se encarrega desta tarefa.

Outras funções desempanhadas pelo CONSTRUTOR são:

- Análise léxica da descrição sintática : préprocessamento da descrição sintática para poupar aos algoritmos de análise sintática o trabalho de procurar símbolos elementares da descrição sintática. A sub-rotina ANALEXCON destina-se a este fim.

- Deteção e manipulação de erros: verificação da correção léxica e sintática da descrição da linguagem, geração das mensagens apropriadas e recomposição da análise. Mecanismos diversos, inseridos em sub-rotinas e no programa principal

cuidam desta tarefa.

- Verificação da estrutura sintática quanto à viabilidade de execução do compilador gerado: deteção de recursão implícita à esquerda na descrição sintática geral e verificação da tabela de precedência quanto à unicidade de relação de precedência entre dois símbolos terminais. Foram programados algoritmos que fazem estas verificações após (recursão à esquerda) ou durante (relações de precedência) a construção das tabelas.

2.2 OS COMPILADORES GERADOS

Dois algoritmos fixos fazem a análise sintática:

- RECONHECEDOR: algoritmo de análise por descida recursiva dirigido pelas tabelas sintáticas gerais.

- RECOPER : algoritmo de análise por precedência de operadores dirigido pela matriz de precedência de operadores.

A análise semântica dos compiladores fica a cargo de rotinas semânticas programadas pelo usuário do sistema.

Outras funções dos compiladores gerados são:

- Leitura das tabelas sintáticas.

- Análise léxica do programa fonte: a sub-rotina ANALEXFON faz o préprocessamento do programa fonte.

- Deteção e manipulação de erros: na análise "top down" o próprio usuário deve inserir certas modificações na descrição sintática visando a deteção de erros conforme será posteriormente detalhado. Na análise "bottom up" a não existência de relação de precedência indica erros no programa fonte.

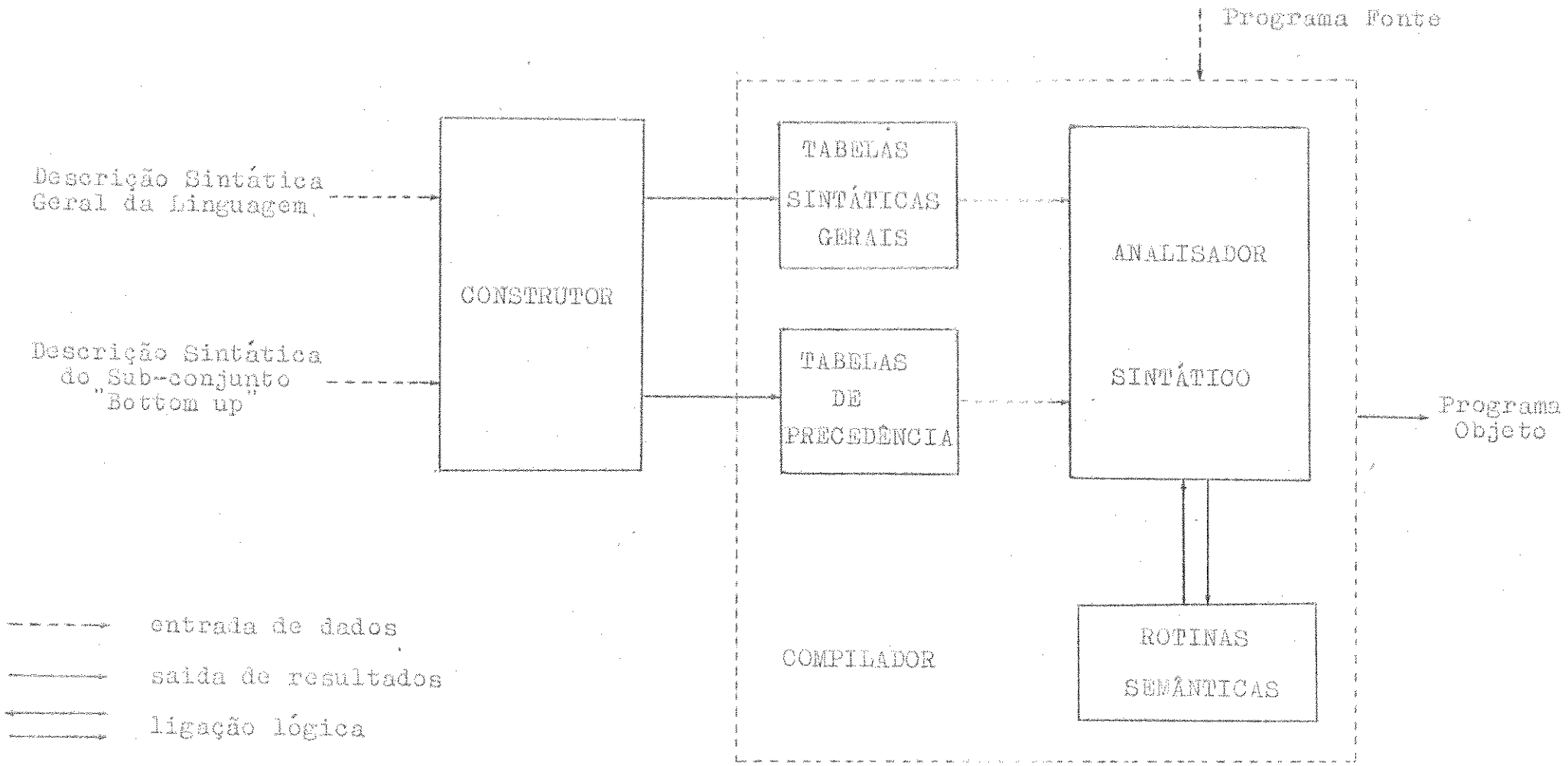

Figura 1. Esquema Geral do Sistema

3. A ESPECIFICAÇÃO SINTÁTICA

Neste capítulo descrevemos as linguagens utilizadas para especificar a sintaxe das linguagens aceitas pelo CONSTRUTOR. Tais linguagens são normalmente conhecidas como metalinguagens sintáticas.

3.1 GRAMÁTICAS DE CONTEXTO LIVRE E BNF

No sentido usual entendemos gramática como um conjunto de regras que devem ser observadas na formação de frases sintáticamente corretas de uma linguagem.

Na conceituação formal devida a Chomsky [15], define-se uma gramática G como uma quádrupla:

$G = \{V, T, P, S\}$ onde

V : conjunto de símbolos não terminais,
T : conjunto de símbolos terminais,
$S \in V$: símbolo não terminal distinto e
P : conjunto de produções que definem a estrutura da linguagem.

Note-se que na conceituação formal são dadas regras (conjunto P) que, aplicadas sucessivamente a partir do símbolo distinto S, dão origem a cadeias de símbolos terminais e não terminais. Um programa fonte é constituido apenas de símbolos terminais.

A maioria das linguagens de programação pode ser definida por gramáticas de contexto livre, isto é, gramáticas cujas produções devem ter a seguinte forma:

$N \rightarrow x$ onde

N (lado esquerdo da produção) é um não terminal e
x (lado direito da produção) é uma cadeia não vazia de símbolos terminais e/ou não terminais.

Uma forma padrão de notação para gramática de contexto livre é a forma de Backus-Naur, BNF, usada para descrever a sintaxe da maioria das linguagens de programação.

Versões modificadas de BNF são utilizadas para descrever a sintaxe das linguagens aceitas pelo CONSTRUTOR. A notação utilizada difere da notação BNF nos seguintes pontos:

- O término de cada produção é indicado por ;
- Terminais são indicados por caracteres entre apóstrofos.
- Classes sintáticas simples são consideradas terminais especiais (metaterminais) e são representadas da seguinte forma: identificadores <*I> , número <*N> , cadeia <*S> .
- O caracter de alternação em BNF | , é representado pelo caracter / .

3.2 ESPECIFICAÇÃO SINTÁTICA GERAL

A especificação sintática da parte geral de uma linguagem deve ser feita pela versão modificada de BNF cuja sintaxe é descrita no apêndice A .

A notação BNF clássica, embora conveniente, não é apropriada para a maioria dos geradores de compiladores.

O problema principal, em nosso caso (gerador "top down"), é o da recursão à esquerda que pode fazer os analisadores entrarem em "loops" infinitos. Acrescenta-se a êste problema a necessidade de dar nome a todas as classes sintáticas, o que pode tornar a descrição demasiadamente longa. Introdução de fatoração por parentesis e outras modificações contornam, em parte, estes problemas.

3.2.1 CARACTERES DE REPETIÇÃO * , + , ?

Caracteres de repetição (utilizados por Trout e Gaffney) resolvem parte do problema de recursão à esquerda:

* indica zero ou mais ou nais ocorrências do conjunto ao qual se aplica.

Exemplo:

A ::= λ | AB em BNF (λ representa a cadeia nula)

pode ser escrita

A ::= B*; em BNF modificada.

\+ indica uma ou mais ocorrências do conjunto ao qual se aplica

A ::= B|AB em BNF

pode ser escrita

A ::= B+; em BNF modificada

? indica ocorrência opcional do conjunto ao qual se aplica.

Exemplo:

A ::= λ| B em BNF

pode ser escrita

A ::= B?; em BNF modificada.

3.2.2 PARENTESIS

Parentesis simplificam bastante a descrição sintática quando temos alternativas que se iniciam por um mesmo símbolo.

Exemplo:

A ::= BC | BD | BE | FG em BNF

pode ser escrita

A ::= B(C/D/E)/FG; em BNF modificada.

Uma expressão entre parentesis é equivalente a uma definição de uma classe sintática e a ela também podem ser aplicados caracteres de repetição. O próprio CONSTRUTOR se encarrega de dar nome a uma classe sintática cuja definição é a expressão entre parentesis.

3.2.3 METATERMINAIS ESPECIAIS

Os sub-conjuntos "bottom up" são metaterminais para o analisador sintático e como tal devem ser representados na especificação sintática geral. Metaterminais especiais representam, portanto, sub-conjuntos "bottom up" da linguagem e são representados do mesmo modo como representamos metaterminais comuns.

Exemplo: <*A> = expressão aritmética

3.2.4 RESTRIÇÕES À METALINGUAGEM

Recursão explícita à esquerda pode ser evitada por meio dos caracteres de repetição. Mais difícil de se evitar mas de consequência igualmente fatal é a recursão implícita à esquerda.

Exemplo:

```
A ::= B...
B ::= F...
.
.
.
F ::= A...
```

A providência cabível é indicar ao usuário a existência deste tipo de recursão. Para tanto foi incluido no CONSTRUTOR um algoritmo que deteta quais as classes sintáticas envolvidas.

A ordenação das alternativas dentro da definição de uma classe sintática também exerce um papel muito importante na análise, notadamente quando as alternativas possuem porções coincidentes. De um modo geral, as alternativas mais longas devem

ser testadas primeiro.

Por exemplo se tivermos a produção

A ::= '<'/' < ' '=' ; deveríamos escrevê-la

A ::= '<' '='/' < ' ; para que o analisador não se "satisfaça" cedo demais ao encontrar o caracter < seguido de = .

Também é importante a ordenação para se evitar que o analisador retorne frequentemente, o que poderia tornar necessário "desfazer" rotinas semânticas. Um projeto cuidadoso de descrição sintática pode minimizar os retornos do analisador, tornando-o mais eficiente.

3.3 ESPECIFICAÇÃO SINTÁTICA PARA SUB-CONJUNTOS "BOTTOM UP"

Os sub-conjuntos "bottom up" serão descritos utilizando a notação apresentada em 3.1. Tais sub-conjuntos devem constituir gramáticas de precedência de operadores que são sub-conjuntos de gramáticas de contexto livre, caracterizados por:

a) Não possuem produções que tenham dois ou mais não terminais adjacentes no lado direito;
b) Entre dois terminais não existe mais do que uma relação de precedência.

As relaçoes de precedência entre dois terminais (Floyd [2]) R e S de uma gramática de operadores são definidas do seguinte modo:

1. R ≐ S se e sòmente se existe uma produção U ::= ...RS... ou U ::= ...RVS..., V não terminal.

2. R ⋖ S se e sòmente se existe uma produção U ::= ...RW... e existem produções tais que a partir de W podemos obter S... ou VS..., V não terminal.

3. R ⋗ S se e sòmente se existe uma produção U ::= ...WS... e

existem produções tais que a partir de W podemos obter ...R ou ...RV, V não terminal.

Para que possam dar origem a compiladores executáveis os sub-conjuntos "bottom up", além de serem gramáticas de precedência de operadores, não podem ter produções com lados direitos iguais.

3.4 INSERÇÃO DE SEMÂNTICA

A mesma notação serve para inserir semântica tanto na descrição sintática geral como na descrição de sub-conjuntos "bottom up".

Três tipos de chamadas de rotinas semânticas podem aparecer dentro de uma especificação sintática: @T, @E e @S, seguidas do número da rotina semântica.

a) @S : são chamadas colocadas opcionalmente ao fim das alternativas de produções.

b) @T : são chamadas de rotinas que verificam a compatibilidade de tipos de variáveis e são inseridas logo após identificadores (<*I>).

c) @E : são chamadas de rotinas especiais, utilizadas somente na descrição sintática geral, inseridas em qualquer ponto do lado direito de uma produção. São para situações em que é necessário efetuar ações semânticas antes de ser encontrada uma alternativa completa.

Exemplos:

```
<TERMO> ::= <PRIMÁRIO> / <TERMO> '*'<PRIMÁRIO>
                @S 10;
<PRIMÁRIO> ::= <*I> @T 11 @S 12;

<LISTIP> ::= <*I> @E 7 ( ',' <*I> @S 8 );
```

Para efeito de descrição sintática as ações @E e

@T têm o valor de um elemento da produção.

3.5 RECUPERAÇÃO DE ERROS

Mecanismos de deteção e recuperação de erros na parte geral da linguagem podem ser inseridos na própria especificação sintática. Basta ter-se o conhecimento de onde e como podem surgir erros e inserir nos locais apropriados alternativas com rotinas especiais que cuidarão adequadamente do erro detetado.

Exemplo:

1) <AP VA PARA> ::= 'VA' ('PARA'/@E 51)
(<*I>/@E 52) @S 10;

- Quando falhasse a busca da palavra reservada PARA ou de um identificador as rotinas 51 ou 52 se encarregariam das mensagens de erro apropriadas e fariam com que a análise prosseguisse procurando na cadeia de entrada ; ou FIM. A partir do ponto em que fosse encontrado um erro as rotinas semânticas inseridas por S não deveriam mais gerar código.

2) <OUTRAFIM> ::= ';'/@E 21 (<AFIRM> / ';'/@E 39);

- Num nível superior de análise a única mensagem emitida seria a de que foi encontrado erro em determinado ponto do programa fonte e a análise prosseguiria após a rotina 39 ignorar símbolos até encontrar ; ou FIM.

Erros nos sub-conjuntos "bottom up" não necessitam de tais artifícios uma vez que o próprio algoritmo de análise pode detetá-los e providenciar a recuperação.

4. TABELAS GERADAS PELO CONSTRUTOR

4.1 A TABELA SINTÁTICA GERAL

A tabela sintática geral (TOPDWN) é, na realidade, um conjunto de 4 tabelas ligadas:

1) Tabela de Não Terminais ou Classes Sintáticas
2) Tabela de Definição Sintática
3) Tabela de Terminais
4) Tabela de Palavras Reservadas

Na Figura 2 estão esquematizadas as ligações entre as tabelas.

4.1.1 TABELA DE NÃO TERMINAIS

Nesta tabela associado ao nome de cada não terminal é colocado também o apontador (APTABSINT) para a primeira alternativa de sua definição na tabela de definição sintática. Para cada expressão entre parentesis é colocada uma entrada nesta tabela do não terminal GERADO com o apontador correspondente para sua definição.

[illegible]	[illegible]

4.1.2 TABELA DE DEFINIÇÃO SINTÁTICA

Dois tipos de entradas são possíveis nesta tabela: cabeça de alternativa e entrada comum.

A cabeça de uma alternativa é composta de três elementos:

a) Apontador (TABSINT) para a próxima entrada da tabela; no caso corresponde ao primeiro ítem da alternativa

b) Número da rotina semântica a ser executada se forem encontrados todos os ítens da alternativa; se não existir rotina semântica o número é

zero.

c) Apontador para a próxima alternativa da definição.

TABSINT	SEM	APALTERN

Uma entrada comum também tem três elementos:

a) Apontador para próxima entrada da tabela: aponta o item seguinte da alternativa presente; se o item atual for o último da alternativa o apontador contém zero.

b) Tipo do item (TIPO)

Um número é atribuido a cada tipo de item:

0 : não terminal sem caracter de repetição

1 : não terminal seguido de ?

2 : não terminal seguido de +

3 : não terminal seguido de *

4 : terminais .

5 : <*I>

6 : <*N>

7 : <*S>

8 : <*A> - expressão aritmética

9 : palavras reservadas

11 : @T

12 : @E

c) Dependendo do tipo este elemento pode conter:

0, 1, 2, 3 - apontador para a tabela de não terminais.

4 - apontador para a tabela de terminais

5, 6, 7, 8 - zero

9 - apontador para a tabela de palavras reservadas

11, 12 - número da rotina semântica.

TABSINT	TIPO	APELEMENTO

4.1.3 TABELA DE TERMINAIS

TERM contém as palavras reservadas na ordem em que foram encontradas durante a geração, sem repetição.

4.1.4 TABELA DE PALAVRAS RESERVADAS

PALARES contém as palavras reservadas na ordem em que foram encontradas durante a geração, sem repetição. Palavras reservadas são identificadores que têm um significado especial na linguagem e não devem ser usadas para variáveis do programa fonte.

4.2 A MATRIZ DE PRECEDÊNCIA DE OPERADORES

A matriz de precedência de operadores (MATPREC) guarda as relações de precedência entre pares de terminais de seguinte modo:

precedência igual: =
precedência menor: <
precedência maior: >
ausência de precedência: branco

Os terminais "bottom up" são armazenados em ordem crescente numa tabela para facilitar o processo de busca durante a análise e MATPREC é ordenada seguindo a mesma sequência dos terminais.

Além disso, a cada terminal é associado um par de números correspondentes às rotinas semânticas @S e @T. Zeros indicam que não há rotinas semânticas a serem executadas.

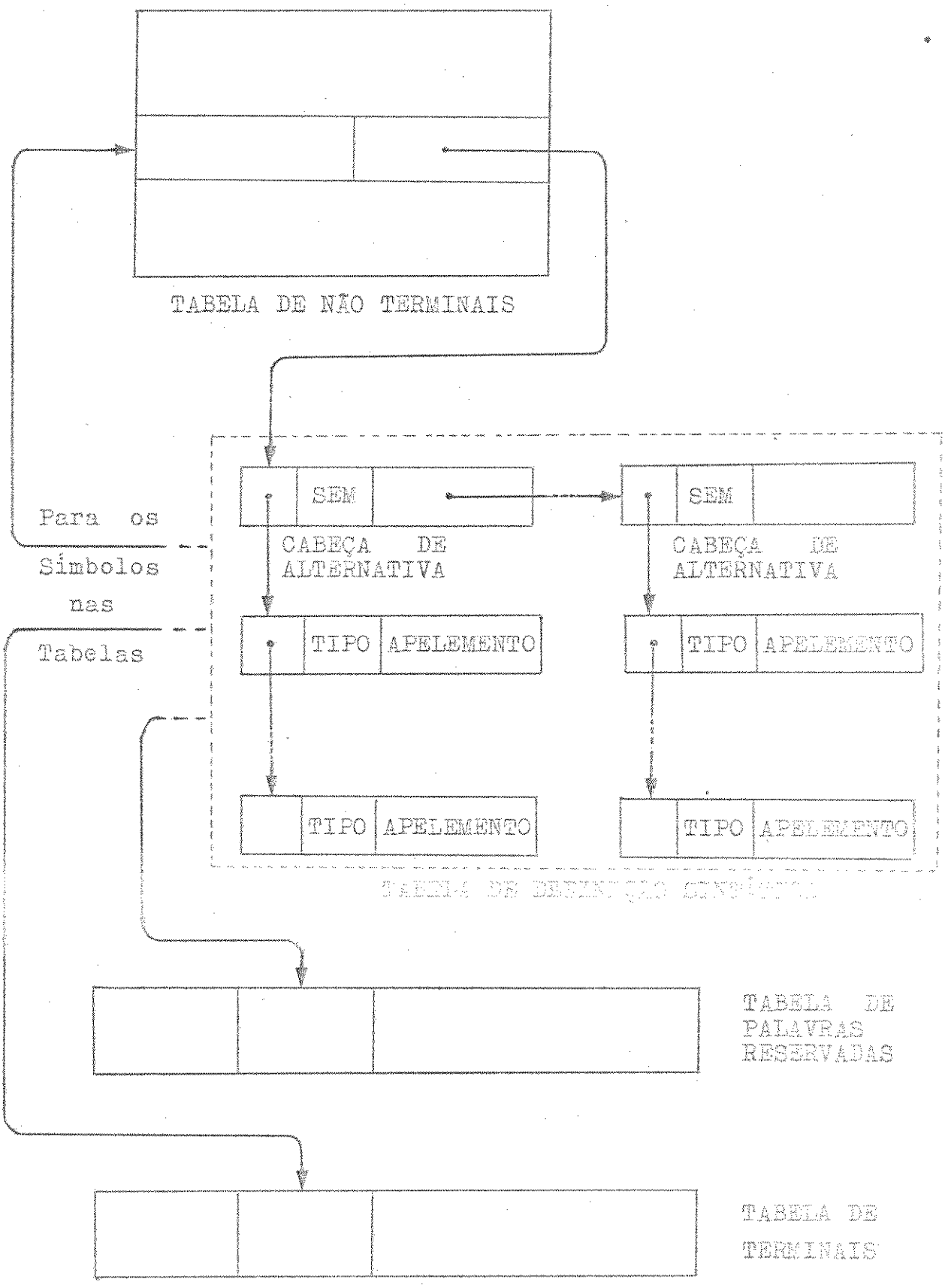

Figura 2. A Tabela Sintática Geral

5. O CONSTRUTOR

O CONSTRUTOR foi programado como um programa principal que constrói a tabela sintática geral, uma sub-rotina (ANALEXCON) de análise léxica, uma sub-rotina (CONSTRUTOPER) para geração da matriz de precedência e várias sub-rotinas auxiliares para tarefas diversas como recuperação de erros, leitura e classificação de caracteres, etc. Na Figura 3 mostramos o esquema geral do CONSTRUTOR.

A geração das tabelas segue a seguinte sequência:

a) Análise e geração da tabela sintática geral: toda vez que o programa principal necessita de um novo símbolo faz-se uma chamada de ANALEXCON;

b) Verificação de recursão à esquerda.

c) Armazenagem da tabela sintática em disco.

d) Análise e geração da matriz de precedência: ANALEXCON é chamada sempre que houver necessidade de um novo símbolo. Durante a própria geração verifica-se a correção da matriz de precedência.

e) Armazenagem da matriz de precedência.

O programa principal controla a, b, e c e CONSTRUTOPER controla d, e e . Após realizada sua tarefa, CONSTRUTOPER entrega o ccntrole ao programa principal.

5.1 ANALISADOR LÉXICO

ANALEXCON faz a análise léxica da descrição sintática passando para o programa principal ou para CONSTRUTOPER os seguintes símbolos:

a) Não terminais: <letras e brancos>

b) Terminais : 'cadeia de caracteres não com

çando por letra'

c) Números

d) Identificadores

e) Palavras reservadas: 'cadeia de letras'

f) <*I> , <*S> , <*N> , <*A> , @T , @S e @E

g) *, +, ?, (,) , ::= e ;

Aos símbolos de a,b,c,d,e e f são atribuidos números que facilitam a tarefa de construção das tabelas.

Para CONSTRUTOPER palavras reservadas (c) poderiam também ser consideradas terminais desde que não houvesse coincidência com palavras reservadas da descrição sintática geral. Os símbolos *, + , ? , (e) não têm significado especial para CONSTRUTOPER, a não ser como terminais (devendo neste caso aparecer entre apostrofos). Os símbolos <*I> , <*S> e <*N> são símbolos terminais para precedência de operadores.

5.2 CONSTRUÇÃO DA TABELA SINTÁTICA

O algorítmo de construção da tabela sintática geral examina cada produção procurando verificar a sua correção sintática e gerando a tabela sintática.

A primeira ocorrência de um não terminal, terminal ou palavra reservada provoca sua entrada na tabela correspondente.

A ocorrência de um não terminal como lado esquerdo de uma produção ocasiona a atualização de seu apontador (APTABSINT) para a sua definição na tabela sintática. Se o apontador contiver um número diferente de zero o algorítmo dá a indicação de erro. Qualquer símbolo que não seja não terminal como lado esquerdo de uma produção é detetado como

erro.

Ao iniciar a procura de símbolos do lado direito de uma produção o algorítmo determina uma cabeça de alternativa na tabela sintática e guarda a sua ocorrência em CAB (CABGER se a produção corresponde a uma expressão entre parentesis).

Segue-se um resumo das ações do algorítmo quando encontra os seguintes símbolos do lado direito:

a) Não terminal : é lido o próximo símbolo para determinação do tipo e atualiza-se, se necessário, a tabela de não terminais, atualizam-se os apontadores da tabela sintática (TABSINT e APELEMENTO) e coloca-se o tipo do não terminal (0,1,2, ou 3).

b) Terminal ou palavra reservada : atualizam-se a tabela correspondente (se necessário) e os apontadores TABSINT e APELEMENTO; coloca-se também o tipo correspondente (- ou +).

c) @T, @E : lê-se o próximo símtolo, que deve ser um número, e este é colocado na tabela sintática (APELEMENTO); atualiza-se TABSINT e coloca-se o tipo do símbolo (11 ou 12).

d) (: Procede-se como se tivéssemos uma nova produção introduzindo o não terminal GERADO na tabela de não terminais; guarda-se o valor desta ocorrência em GERSINT e atualiza-se TABSINT; a informação de que ocorreu (é guardada

(PAR ← TRUE).

e)) : guarda-se a informação de que este é um não terminal GERADO, atribui-se tipo 0 ao mesmo (nota-se que sua ocorrência foi guardada em GERSINT) e procede-se como em b.

f) / : atualiza-se a cabeça da alternativa corrente (CAB ou CABGER) e coloca-se zero no apontador TABSINT da última entrada na tabela sintática.

g) <*I> , <*S> , <*N> ou <*A> : atualiza-se o apontador TABSINT e é colocado o tipo correspondente (4,5,6,7 ou 8) na tabela sintática.

h) @S : lê-se próximo símbolo, que deve ser um número, e coloca-se este na cabeça de alternativa corrente (CAB ou CABSIN).

i) ; : coloca-se zero no apontador TABSINT da última entrada da tabela sintática e procura-se nova produção.

Sempre que há novas entradas nas tabelas são incrementados os índices correspondentes para indicar as próximas posições livres.

Como se pode notar, este algoritmo não permite parentesis embutidos na descrição sintática geral. Pequenas modificações no mesmo seriam suficientes para permiti-lo.

O algoritmo inclue também verificação de recursão à esquerda: construidas as tabelas, estas são explorada e caso haja recursão à esquerda, os não terminais envolvidos são

indicados ao usuário.

Erros durante a construção das tabelas ou a existência de recursão à esquerda determinam a atribuição do valor FALSE à variável booleana CONDCOMP, com a consequente mensagem ao usuário de que o compilador produzido não é executável.

5.3. CONSTRUÇÃO DA MATRIZ DE PRECEDÊNCIA

5.3.1 RELAÇÕES ENTRE SÍMBOLOS DE UMA LINGUAGEM

A matriz de precedência (MATPREC) é construída a partir das matrizes de relações FIRST, FIRSTTERM, LAST, LASTTERM, $\doteq$ e $\lessdot$ definidas em Gries.

Estas relações são definidas como se segue:

1) A FIRST B se

A ::= B ... onde A é não terminal e B é terminal ou não terminal.

2) A FIRSTTERM b se

A ::= b... ou A ::= Cb... onde

A e C são não terminais e b é terminal

3) A LAST B se

A ::= ...B onde

A é não terminal e B é terminal ou não terminal

4) A LASTTERM b se

A ::= ...b ou A ::= ...bC onde

A e C são não terminais e b é terminal

5) A $\doteq$ B se

C ::= ...AB... onde

C é não terminal e B e A são terminais ou não

terminais

6) a $\doteq$ b se
C ::= ...ab... ou C::= ...aDb...
onde C e D são não terminais e a e b são terminais.

As relações entre todos os símbolos de uma gramática podem ser representadas por matrizes booleanas, onde , para cada elemento das matrizes, TRUE corresponde a existência da relação e FALSE à inexistência.

As matrizes são as seguintes:

PRI para a relação FIRST;

PRITERM para FIRSTTERM;

ULT para LAST;

ULTTERM para LASTTERM;

ADJA para $\lessdot$ e

TIG para $\doteq$

De Gries ainda tem-se que:

a) Matriz de precedência menor (PREMENOR) pode ser calculada por:

PREMENOR = (ADJA) (PRI$*$) (PRITERM)

b) Matriz de precedência maior (PREMAIOR) por:

PREMAIOR = TRANSPOSTA ((ULT$*$)(ULTERM)) (ADJA)

c) Matriz de precedência igual é a matriz TIG.

Nas expressões acima temos:

$$PRI* = (I) + (PRI) + (PRI)^2 + \ldots + (PRI)^n$$

$$ULT* = (I) + (ULT) + (ULT)^2 + \ldots + (ULT)^n$$

onde I é a matriz identidade e n é a dimensão das matrizes.

5.3.2 CONSTRUÇAO DAS MATRIZES BOOLEANAS E DE MATPREC

Na descrição que se segue, os termos linguagem, produção, terminal, não terminal, etc. referem-se sòmente ao sub-conjunto "bottom up".

CONSTRUTOPER examina cada produção da linguagem, verifica sua correção sintática e contrói as matrizes PRI, PRITERM, ULT, ULTTERM, ADJA e TIG.

É construida uma tabela provisória geral de todos os símbolos da linguagem, PALAVRA, que auxiliará a construção das matrizes booleanas e de MATPREC; TERMOP é a tabela de terminais.

A primeira ocorrência de um símbolo terminal ou não terminal provoca a sua entrada em PALAVRA e em TERMOP (terminal).

São definidas as seguintes variáveis:

IDLEM : posição do símbolo corrente em PALAVRA

ITERMOP : posição do terminal corrente em TERMOP

PRODU : posição em PALAVRA do lado esquerdo de uma produção.

INICT : variável booleana, TRUE se ainda não foi encontrado o primeiro terminal do lado direito de uma produção.

INICN : variável booleana, TRUE se ainda não foi encontrado o primeiro símbolo do lado direito.

ULTNT : variável booleana, TRUE se o último símbolo analisado do lado direito tiver sido

um não terminal.

IADJA e IULT : posição em PALAVRA do último símbolo analisado.

IULTTERM : posição em PALAVRA do último terminal analisado.

ITIG : posição em TERMOP do último terminal analisado.

O algorítmo de construção ao encontrar o lado esquerdo (não terminal) de uma produção atualiza PALAVRA (se for necessário) e guarda em PRODU a posição do não terminal na mesma.

Antes de iniciar a busca de símbolos do lado direito são inicializadas as variáveis booleanas:

INICN ⟵ TRUE; INICT ⟵ TRUE;
ULTNT ⟵ FALSE

Segue-se um resumo das ações do algorítmo ao encontrar os seguintes símbolos do lado direito:

a) Não terminal : se ULTNT = TRUE então erro senão atualiza PALAVRA (se necessário) e coloca em IELEM a posição do não terminal em PALAVRA;

ULTNT ⟵ TRUE;

Se INICN = TRUE então
PRI [PRODU, IELEM] ⟵ TRUE,
INICN ⟵ FALSE
senão ADJA [IADJA, IELEM] ⟵ TRUE;
IADJA ⟵ IELEM, IULT ⟵ IELEM.

b) Terminal : atualiza PALAVRA e TERMOP;
IELEM ⟵ posição em PALAVRA,
ITERMOP ⟵ posição em TERMOP;

ULTNT ← FALSE;
Se INICN = TRUE então
PRI [PRODU,IELEM] ← TRUE,
INICN ← FALSE
senão ADJA [IADJA,IELEM] ← TRUE;
se INICT = TRUE então
PRITERM [PRODU, IELEM] ← TRUE,
INICT ← FALSE senão
TIG [ITIG,ITERMOP] ← TRUE;
ITIG ← ITERMOP; IULTTERM ← IELEM
IADJA ← IELEM; IULT ← IELEM.

c) @T, @S : lê próximo símbolo, que deve ser um número, e coloca-o na posição correspondente ao último terminal lido na tabela de rotinas semânticas.

d) / : ULT [PRODU, IELEM] ← TRUE;
ULTTERM [PRODU,IULTTERM] ← TRUE;
INICT ← TRUE; INICN ← TRUE;
ULTNT ← FALSE.

e) ; : ULT [PRODU, IELEM] ← TRUE;
ULTTERM [PRODU, ITERMOP] ← TRUE;
procura próxima produção.

Construidas as matrizes PRI, PRITERM, ADJA, ULTTERM, ULT e TIG são determinadas as matrizes ULT* e PRI* e em seguida as matrizes PREMENOR e PREMAIOR.

No processo de construção de MATPREC, a partir das matrizes PREMENOR, PREMAIOR e TIG, verifica-se se não há mais do que uma relação de precedência entre dois terminais.

Se tudo estiver correto ordena-se TERMOP e MATPREC, pela ordem crescente dos terminais.

Erros durante a construção de MATPREC determinam a atribuição do valor FALSE à variável boolena CONDCOMPOP e a emissão da mensagem de erro correspondente.

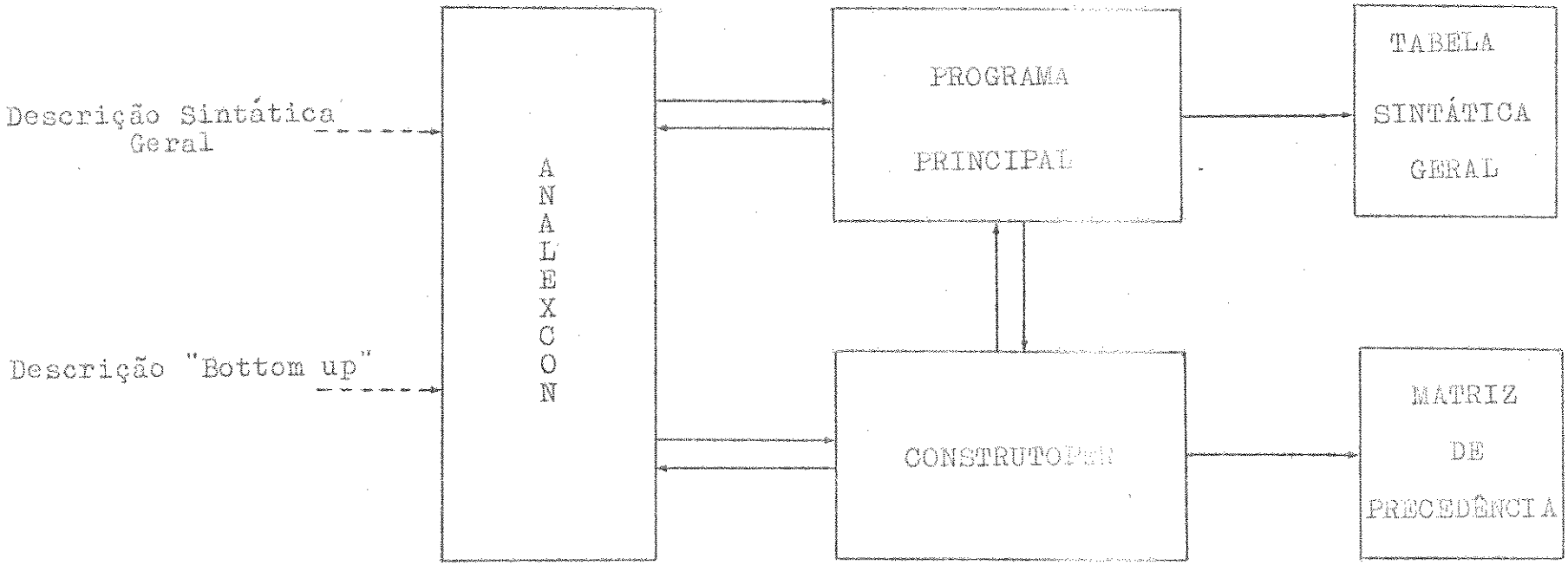

Figura 3. O Construtor

6. O COMPILADOR

O compilador foi implementado como um programa principal que faz a leitura das tabelas (TOPDWN e EXPRAR), uma sub-rotina (ANALEXFON) de análise léxica, uma sub-rotina (RECONHECEDOR) de análise sintática geral, uma sub-rotina (RECOPER) de análise sintática "bottom up", rotinas semânticas e várias sub-rotinas auxiliares para busca de terminais operadores, busca e inserção de identificadores, etc.

O esquema geral do compilador pode ser visto na Figura 4.

Devido ao problema de retorno na análise sintática "top down" o programa fonte todo é analisado lexicamente e armazenado na memória antes de se iniciar a análise sintática. Os valores iniciais das variáveis são dados antes de se fazer a chamada inicial de RECONHECEDOR.

6.1. ANALISADOR LÉXICO

Os símbolos procurados por ANALEXFON são:

Identificadores - tipo 1

Números inteiros - tipo 2

Palavras especiais ('letras') - tipo 3

Cadeias ("caracteres") - tipo 4

Outros símbolos simples - tipo 5

Aspas (") e ; como caracteres da cadeia devem aparecer duplicados. A limitação no comprimento dos símbolos compostos é definida pela implementação.

O uso de palavras especiais exige que os identificadores com significado especial na linguagem se apresentem sempre entre apostrofos no programa fonte.

Na implementação do compilador preferiu-se palavras reservadas. Estas são procuradas como identificadores por ANALEXFON; este fato implica em se tomar certos cuidados na descrição sintática geral: as alternativas que se iniciam por palavras reservadas devem ser testadas antes das que se iniciam por identificadores. Uma busca preliminar na tabela de palavras reservadas ou a opção de palavras especiais contornariam este problema.

O próprio usuário pode definir símbolos especiais programando convenientemente um analisador léxico apropriado.

6.2 ANALISADOR SINTÁTICO GERAL

RECONHECEDOR foi programado como um algorítimo de descida recursiva. Dados os valores iniciais referentes ao primeiro não terminal (símbolo distinto) o algorítmo inicia a análise do programa fonte e continua-a até terminar a cadeia de símbolos do programa fonte. Ao procurar um não terminal do lado direito de uma produção o algorítmo chama a si mesmo, guardando antes as informações referentes ao não terminal que estava sendo procurado.

O algorítmo faz uso da variável booleana CERTO (TRUE indica sucesso e FALSE insucesso) e dos tipos dos itens nos seguintes passos:

1 : procura a definição indicada por K (apontador para a tabela sintática); guarda a posição do programa fonte;

2 : guarda a cabeça da alternativa;

3 : guarda a entrada do item na tabela sintática;

4 : desvia para a rotina correspondente ao tipo do item;

tipo 0 : chama RECONHECEDOR; vai para TESTE;

tipo 1 : chama RECONHECEDOR; CERTO ← TRUE; vai para 5;

tipo 2 : chama RECONHECEDOR; se CERTO = TRUE vai para 6 senão vai para tipo 3;

tipo 3 : chama RECONHECEDOR; se CERTO = TRUE vai para tipo 3 senão CERTO ← TRUE e vai para 5;

tipo 4 : verifica o tipo do símbolo; se coincide com procurado CERTO ← TRUE e incrementa posição do programa fonte senão CERTO ← FALSE; vai para TESTE;

tipo 5 : idem;

tipo 6 : idem;

tipo 7 : idem;

tipo 9 : idem;

tipo 8 : chama RECOPER; vai para TESTE;

tipo 11 ou tipo 12 : chama ROTINA; CERTO ← TRUE; vai para TESTE;

TESTE : se CERTO = TRUE vai para 5 senão vai para 6.

5 : se existe próximo item na alternativa então vai para 3 senão : se existe rotina semântica chama ROTINA e retorna senão retorna;

6 : restaura posição do programa fonte; se não existe próxima alternativa retorna senão vai para 2.

Note-se que sempre que um não terminal surge como item de uma alternativa o RECONHECEDOR é chamado para tentar reconhecê-lo. Se um terminal (4,5,6,7, ou 9) é encontrado como item é feita uma verificação do símbolo corrente do programa fonte. Se o item é uma expressão aritmética (tipo 8) a sub-rotina RECOPER é chamada. Se o item é uma rotina semântica esta é executada.

Se o reconhecimento de algum item falha, o analisador descontinua a análise dentro desta alternativa, restaura o apontador do programa fonte para sua posição no início da chamada e tenta a próxima alternativa. Se todas as alternativas falham, o não terminal é considerado não encontrado e a análise retorna com esta indicação (CERTO = FALSE) ao ponto em que o não terminal foi procurado.

Se todos os itens de uma alternativa foram reconhecidos, a rotina semântica (opcional) é executada e o não terminal é considerado reconhecido.

6.3. ANALISADOR SINTÁTICO POR PRECEDÊNCIA DE OPERADORES

RECOPER foi programado como o algorítmo clássico de análise por precedência de operadores com:

STACK : pilha contendo símbolos lidos

I e J : índices de contrôle

IPROG : apontador para o símbolo corrente do programa fonte

VAR : variável auxiliar

ITEMPANT : variável auxiliar

PROG : matriz contendo o programa fonte

O algorítmo tem os seguintes passos:

1 : STACK [1] ← @ ; I ← 1 ; IPROG decrementado de 1 em relação ao valor no instante da chamada de RECOPER;

2 : IPROG ← IPROG + 1; VAR ← PROG [IPROG]

3 : Se STACK [I] é terminal então J ← I senão J ← I - 1;

4 : STACK [J] > VAR ?
sim : I ← I + 1; STACK [I] ← VAR; vai para 2

Não : vai para 5;

5 : ITEMPANT ← STACK [J] ; J ← J - 1;
Se STACK [J] é terminal então vai para 6 senão
J ← J - 1 e vai para 6;

6 : STACK [J] < ITEMPANT ?
Não : vai para 5;
Sim : vai para 7;

7 : STACK [J+1] ... STACK [I] é o conjunto que corresponde a uma classe sintática; chama ROTINA para processá-lo;
I ← J + 1; STACK [I] ← nome do não terminal;
Se I = 2 e VAR = # então pára senão vai para 4.

Erros serão detetados quando for encontrado branco na matriz de precedência. Rotinas de teste podem ser chamadas logo que for lido um terminal (a cada terminal são associados dois números de rotinas semânticas conforme já foi visto). No passo 7 a rotina a ser executada é a correspondente ao último terminal do conjunto STACK [J + 1] ... STACK [I].

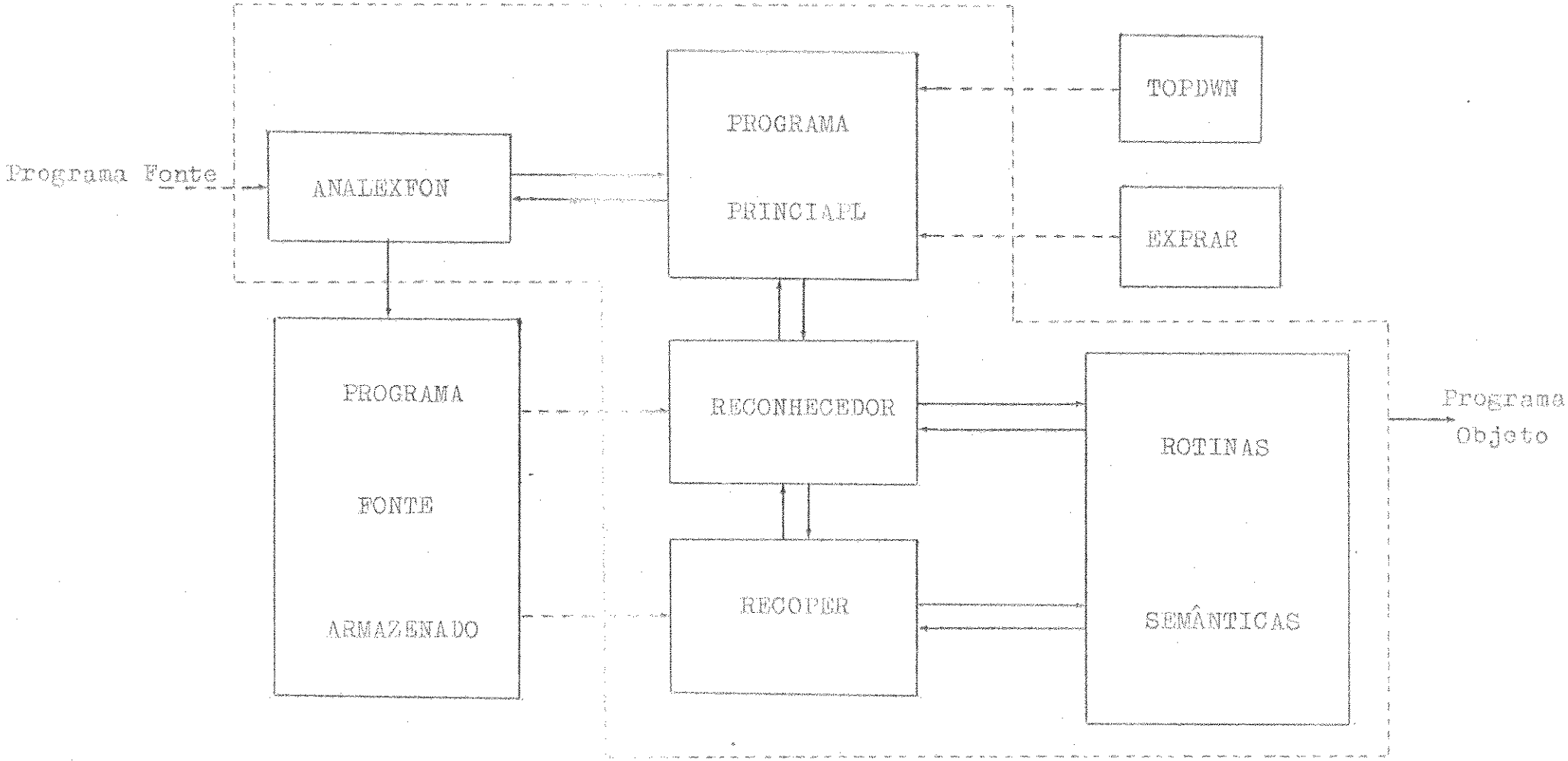

Figura 4. O Compilador

7. ESPECIFICAÇÃO SEMÂNTICA

Conforme foi solientado não existe uma especificação semântica formal para o sistema. O próprio usuário deve programar suas rotinas semânticase inserir as suas chamadas na descrição sintática.

Não há diferença alguma quanto ao modo como são chamadas os três tipos de rotinas semânticas pelos analisadores: na presente implementação é fornecido o número da rotina semântica à sub-rotina ROTINA.

7.1 LIGAÇÃO ENTRE OS ANALISADORES E A SEMÂNTICA

As rotinas semânticas precisam ter acesso às seguintes variáveis:

PROG : programa fonte analisado lexicamente

IPROG : apontador para o símbolo corrente do programa fonte

NUMTEMP : variável que armazena temporariamente os números do programa

IDENTEMP : variável que armazena temporariamente os identificadores do programa

CADTEMP : variável que armazena temporariamente as cadeias do programa

TERMTEMP : variável que armazena temporariamente os terminais do programa

STACK : pilha da análise por precedência de operadores

CERTO : variável booleana de sucesso ou insucesso da análise

As variáveis PROG e IPROG são necessárias para recuperação de erros no programa fonte conforme foi visto no capítulo 3. As outras variáveis são necessárias para a geração de có

digo e construção de tabelas do programa objeto.

7.2 A SUB-ROTINA SEMÂNTICA

Todas as rotinas semânticas foram incorporadas numa só sub-rotina (ROTINA) à qual deve ser dado apenas o número da rotina desejada. Assim a chamada ROTINA(N) fará com que seja executada a rotina semântica de número N. Uma vez executada a rotina semântica o contrôle volta ao analisador sintático após o ponto de chamada de ROTINA.

8. EXEMPLO DE UTILIZAÇÃO

O sistema foi utilizado para construir um compilador para um sub-conjuto de ALGOL (DEMAGL) cuja descrição sintática é dada no Apêndice B.

As tabelas geradas pelo CONSTRUTOR estão nos Apêndices C e D. Apenas um sub-conjunto "bottom up", expressão aritmética, foi separado da descrição sintática e não foi implementado nenhum mecanismo completo de recuperação de erros para a parte geral da linguagem.

As rotinas semânticas foram programadas para fornecer como código objeto um programa em ALGOL do PDP-10. Implementou-se também uma sub-rotina de busca e inserção em uma tabela de identificadores organizada como uma árvore binária. A geração de código para expressão aritmética foi feita através do mecanismo de pilha polonesa (operandos e operadores).

9. DISCUSSÃO FINAL

O sistema gerador de compiladores aqui descrito foi implementado numa linguagem de alto nível, ALGOL, no sistema PDP-10 do Centro de Computação da Universidade Estadual de Campinas.

O CONSTRUTOR foi programado num só bloco: programa principal, CONSTRUTOPER e outras sub-rotinas, que devem ser compilados em conjunto; o mesmo feito com o compilador: programa principal, RECONHECEDOR, RECOPER, ROTINA e demais sub-rotinas.

Foi dada à descrição sintática o seguinte formato:

```
<TITULO> ::= <nome da linguagem>
{  Descrição "top down"
END

<nome do sub-conjunto>
{  Descrição "bottom up"
END
```

Uma limitação muito importante do método "top down" é a dificuldade de manipulação de erros por não ser possível ao algoritmo de análise mecanismos adequados de detecção e recuperação. Fica pois a manipulação de erros a cargo do usuário através de rotinas do tipo E. Para auxiliá-lo nesta tarefa, o sistema torna disponível uma variável, IPROGMAX, apontando para o símbolo mais interno analisado; isto dá uma indicação de até onde o analisador conseguiu chegar, no programa fonte. Outras variáveis que podem auxiliar na emissão de mensagens de erro são NCOLUNA e NCARTAO, para o caso em que o programa fonte esteja contido originalmente em cartões.

A necessidade de se armazenar o programa fonte todo na memória, para possibilitar retorno em caso de caminhos falsos, restringe o comprimento dos programas que podem ser analisados. A introdução de um símbolo marcador especial na descrição sintática aliviaria em parte este problema. O usuário inseriria este marcador em certos pontos da descrição

sintática para indicar que não haveria retornos para símbolos do programa fonte anteriores a estes pontos. Deste modo o analisador poderia manter apenas parte do programa fonte na memória. Tentativas de retornar aquém dos pontos indicados seriam detetados como erros.

A ordenação das alternativas das produções desempenha um papel muito importante na eficiência do analisador "top down". Um estudo cuidadoso da descrição sintática pode levar a um analisador que minimize o número de retornos.

O sistema é razoavelmente flexível:

- Fornecendo-se-lhe a descrição sintática com um formato adequado, pode gerar analisadores exclusivamente "top down" ou exclusivamente "bottom up".

- Embora tenha sido programado para um só conjunto "bottom up", pequenas modificações seriam suficientes para permitir vários conjuntos "bottom up".

Alguns melhoramentos poderiam ser introduzidos no sistema:

- Implementação de um construtor de analisadores léxicos, permitindo ao usuário escolher os símbolos metaterminais de sua linguagem.

- Um algoritmo para deteção de não terminais inúteis (isto é, que não têm influência alguma na linguagem gerada pela gramática, ou por não figurarem no lado esquerdo de nenhuma produção ou por serem simples renominações de uma classe sintática) : teria utilidade na fase de depuração da sintáxe de uma linguagem.

- Utilização de sub-rotinas compiláveis separadamente do programa principal ("external procedures") : isto daria grande liberdade ao usuário do sistema. Por exemplo, na programação de rotinas semânticas, estas poderiam ser programadas e compiladas separadamente, sem necessidade de compilar o analisador todo.

- Um algoritmo mais eficiente para a geração das matrizes de precedência: o algoritmo implementado é muito lento devido ao fato de ter muitos passos e envolver operações com matrizes.

O CONSTRUTOR, para gerar as tabelas da linguagem utilizada como exemplo (DEMAGL), levou pouco mais de 15 segundos (tempo de execução). Por outro lado o compilador obtido, para um programa fonte com 20 linhas gastou aproximadamente 6 segundos, um tempo razoavelmente longo se comparado com compiladores comerciais mas perfeitamente aceitável para compiladores experimentais e em pesquisa de linguagens em geral.

APÊNDICE A - SINTAXE DA BNF MODIFICADA

```
(1)  <SINTAXE>           ::= <PRODUCAO>+ 'END' ;
(2)  <PRODUCAO>          ::= <CLASSE> '::=' <DEFINICAO> ';' ;
(3)  <CLASSE>            ::= '<' <NOME DE CLASSE> '>' ;
(4)  <NOME DE CLASSE>    ::= <*I> ;
(5)  <DEFINICAO>         ::= <ALTERNATIVA> ('/' <ALTERNATIVA>)* ;
(6)  <ALTERNATIVA>       ::= <ITEM>+ <ROTINA S>? ;
(7)  <ITEM>              ::= (<CLASSE> / '(' <DEFINICAO> ')' )
                             <REPET>? / <ROTINA T> / <ROTINA E> /
                             <TERMINAL> ;
(8)  <ROTINA S>          ::= '@S' <*N> ;
(9)  <ROTINA T>          ::= '@T' <*N> ;
(10) <ROTINA E>          ::= '@E' <*N> ;
(11) <REPET>             ::= '*' / '?' / '+' ;
(12) <TERMINAL>          ::= '<*I>' / '<*N>' / '<*S>' /
                             '<*' <LETRA> '>' / <*SE> ;
(12) <LETRA>             ::= 'A' / 'B' / ... / 'Z' ;
     END
```

Obs: Note-se que <*SE> corresponde a 'caracteres' e <*S> corresponde a "caracteres".

APÊNDICE B - SINTAXE DE DEMAGL

```
<TITULO>          ::=  <DEMAGL>
<PROGRAMA>        ::=  <BLOCO> @S 1;
<BLOCO>           ::=  'INICIO' @E 2  <PARTDECL>? @E 32 <AFIRM>
                       (';' @E 21 <AFIRM>)* 'FIM' @S 3 ;
<PARTDECL>        ::=  <TIPO> <LISTIP> ';' @E 21
                       (<TIPO><LISTIP> ';' @S 21)*;
<TIPO>            ::=  'INTEIRO' @S 4 / 'ROTULO' @S 5 / 'LOGICO' @S 6;
<LISTIP>          ::=  <*I> @E 7 (',' <*I> @S 8)*;
<AFIRM>           ::=  ( <BLOCO> / <AF VA PARA> / <SE><ENTAO><SENAO>
                       / <LEIA> / <IMPRIMA> / <ESCREVA> / <DESIG> /
                       <*I> ':' @E 9 <AFIRM>)?;
<AF VA PARA>      ::=  'VA' 'PARA' <*I> @S 10;
<LEIA>            ::=  'LEIA' '(' @E 33 <*I> @E 34
                       ( ',' @E 21 <*I> @S 34)* ')' @S 21;
<IMPRIMA>         ::=  'IMPRIMA' '(' @E 35 <*I> @T 34 ',' @E 21
                       <*N> @E 36 ')' @S 21;
<ESCREVA>         ::=  'ESCREVA' '(' @E 37 <*S> @E 38  ')' @S 21 ;
<DESIG>           ::=  <*I> ':' '=' @T 11 <*A> @S 27
                       / <*I> ':' '=' @E 12 <EXPBOOL>;
<SE>              ::=  'SE' @E 13 <EXPBOOL>;
<ENTAO>           ::=  'ENTAO' @E 14 <AFIRM> @S 29;
<SENAO>           ::=  'SENAO' @E 15 <AFIRM> @S 30;
<EXPBOOL>         ::=  <TERBOOL> ( 'OU' @E 16 <TERBOOL>)*;
<TERBOOL>         ::=  <PRIMBOOL> ( 'E' @E 17 <PRIMBOOL>)*;
<PRIMBOOL>        ::=  'VERDADEIRO' @S 19 / 'FALSO' @S 20
                       / '(' @E 21<EXPBOOL> ')' @S 21
                       / <*A> @E 28 ( '<' / '>' / '=') @E 21
                       <*A> @S 28 / <*I> @S 18;
END
 <EXPRESSAO ARITMETICA>
<EXPR>            ::=  '@' <EA> '#' ;
<EA>              ::=  <TERMO> / <EA> '+' <TERMO> @S 22
                       / <EA> '-' <TERMO> @S 22;
<TERMO>           ::=  <PRIMARIO> / <TERMO> '*' <PRIMARIO> @S 22
                       / <TERMO> '/' <PRIMARIO> @S 22;
<PRIMARIO>        ::=  <*I> @T 23 @S 24 / <*N> @S 25
                       / '(' <EA> ')' @S 31;
END
```

APENDICE C

******** TABELAS SINTATICAS ******** COMPILADOR : DEMAGL

** NAO TERMINAIS **

1:	PROGRAMA	1
2:	BLOCO	3
3:	PARTDECL	15
4:	AFIRM	36
5:	GERADO	10
6:	TIPO	25
7:	LISTIF	31
8:	GERADO	21
9:	GERADO	35
10:	GERADO	40
11:	AF VA PARA	61
12:	SE	107
13:	ENTAO	111
14:	SENAO	115
15:	LEIA	68
16:	IMPRIMA	77
17:	ESCREVA	88
18:	DESIG	95
19:	GERADO	72
20:	EXPBOOL	119
21:	TERBOOL	126
22:	GERADO	122
23:	PRIMBOOL	132
24:	GERADO	129
25:	GERADO	140

** TABELA DE DEFINICAO SINTATICA **

1:	2	1	0
2:	0	0	2
3:	4	3	0
4:	5	9	1
5:	6	12	2
6:	7	1	3
7:	8	12	32
8:	9	0	4
9:	14	3	5
10:	11	0	0
11:	12	4	1
12:	13	12	21
13:	0	0	4

14:	0	9	2
15:	16	0	0
16:	17	0	6
17:	18	0	7
18:	19	4	1
19:	20	12	21
20:	0	3	8
21:	22	21	0
22:	23	0	5
23:	24	0	7
24:	0	4	1
25:	26	4	27
26:	0	9	3
27:	28	5	29
28:	0	9	4
29:	30	6	0
30:	0	9	5
31:	32	0	0
32:	33	5	0
33:	34	12	7
34:	0	3	9
35:	36	8	0
36:	37	4	2
37:	0	5	8
38:	39	0	0
39:	0	1	10
40:	41	0	42
41:	0	0	2
42:	43	0	44
43:	0	0	11
44:	45	0	48
45:	46	0	12
46:	47	0	13
47:	0	0	14
48:	49	0	50
49:	0	0	15
50:	52	0	52
51:	0	0	16
52:	53	0	54
53:	0	0	17
54:	55	0	56
55:	0	0	18
56:	57	0	0
57:	58	5	0
58:	59	4	3
59:	60	12	9
60:	0	0	4
61:	62	10	0
62:	63	9	6
63:	64	9	7
64:	0	5	0
65:	66	21	0
66:	67	9	0
67:	68	4	4
68:	69	12	33

69:	70	5	0
70:	71	12	34
71:	72	3	19
72:	73	34	0
73:	74	4	2
74:	75	12	21
75:	0	5	0
76:	0	4	5
77:	78	21	0
78:	79	3	3
79:	80	4	4
80:	81	12	35
81:	82	5	0
82:	83	11	34
83:	84	4	2
84:	85	12	21
85:	86	5	0
86:	87	12	36
87:	0	4	5
88:	89	21	0
89:	90	9	10
90:	91	4	4
91:	92	12	37
92:	93	7	0
93:	94	12	38
94:	0	4	5
95:	96	27	101
96:	97	5	0
97:	98	4	3
98:	99	4	6
99:	100	11	11
100:	0	0	0
101:	102	0	0
102:	103	5	0
103:	104	4	3
104:	105	4	6
105:	106	12	12
106:	0	0	20
107:	108	0	0
108:	109	9	11
109:	110	12	13
110:	0	0	20
111:	112	29	0
112:	113	9	12
113:	114	12	14
114:	0	0	4
115:	116	30	0
116:	117	9	13
117:	118	12	15
118:	0	0	4
119:	120	0	0
120:	121	0	21
121:	0	3	22
122:	123	0	0
123:	124	9	14

124:	125	12	16
125:	0	0	21
126:	127	0	0
127:	128	0	23
128:	0	3	24
129:	130	0	0
130:	131	9	15
131:	132	12	17
132:	0	0	23
133:	134	19	135
134:	0	9	16
135:	136	20	137
136:	0	9	17
137:	138	21	142
138:	139	4	4
139:	140	12	21
140:	141	0	20
141:	0	4	5
142:	143	28	154
143:	144	0	0
144:	145	12	28
145:	152	0	25
146:	147	0	148
147:	0	4	7
148:	149	0	150
149:	0	4	8
150:	151	0	0
151:	0	4	6
152:	153	12	21
153:	0	8	0
154:	155	10	0
155:	0	5	0

** TERMINAIS **

; , : ( ) = < >

** PALAVRAS RESERVADAS **

INICIO	FIM	INTEIRO	ROTULO	LOGICO	VA	PARA
LEIA	IMPRIMA	ESCREVA	SE	ENTAO	SENAO	OU
E	VERDADEIRO	FALSO				

APÊNDICE D

** MATRIZ DE PRECEDENCIA **

	#	(	)	*	+	-	/	<*I>	<*N>	@
#	>		>			>	>			
(		<	=	<	<	<	<	<		<
)	>		>	>	>	>	>			
*	>	<	>	>	>	>	>	<		<
+	>	<	>	<	>	>	<	<		<
-	>	<	>	<	>	>	<	<		<
/	>	<	>	>	>	>	>	<		<
<*I>	>		>	>	>	>	>			
<*N>	>		>	>	>	>	>			
@	=	<		<	<	<	<	<		<

** ROTINAS SEMANTICAS **

0 0 * 0 0 * 31 0 * 22 0 *

22 0 * 22 0 * 22 0 * 24 23 *

25 0 * 0 0 *

B I B L I O G R A F I A

[1] NAUR, P. et alii. - Revised report on the algorithmic language Algol 60. Comm. ACM, 6:1-17, jan. 1963.

[2] FLOYD, R. W. - Syntactic analysis and operator precedence. J. ACM, 10:316-33, jul. 1963.

[3] SAMELSON, K. & BAUER, F. L. - Sequential formula translation. Comm. ACM, 3:76-83, fev. 1960.

[4] GRIES, D. et alii. - The use of transition matrices in compiling. Comm. ACM, 11:26-34, jan. 1968.

[5] FLOYD, R. W. - A descriptive language for symbol manipulation. J. ACM, 8:578-84, out. 1961.

[6] BROOKER, R. A. & MORRIS, D. - An assembly program for a phrase structure language. Comput. J., 3:168-74, 1960.

[7] IRONS, E. T. - A syntax directed compiler for Algol 60. Comm. ACM, 4:51-5, jan. 1961.

[8] CHEATHAM, T. E. & SATTLEY, K. - Syntax directed compiling. Proc. AFIPS S.J.C., 25:31-57, 1964.

[9] TROUT, R. G. - A BNF like language for the description of syntax directed compilers. Urbana, Universidade de Illinois, Departamento de Ciência de Computação, 1969. (Relatório nº300) [Tese de M. S.]

[10] EARLEY, J. - An efficient context-free parsing algorithm. Comm. ACM, 13 (2):94-102, fev. 1970.

[11] GAFFNEY, J. L. - Tacos:a table driven compiler-compiler system Urbana, Universidade de Illinois, Departamento de Ciência de Computação, 1969. (Relatório nº325) [Tese de M. S.]

[12] FELDMAN, J. & GRIES, D. - Translator writing systems. Comm. ACM, 1(2):77-113, fev. 1968.

[13] GRIES, D. - Grammars and languages. In:_____. Compiler Construction for Digital Computers. New York, John Wiley, 1971, p.11-48.

[14] _____. - Other bottom-up recognizers. In:_____. Compiler Construction for Digital Computers. New York, John Wiley, 1971, p.121-53.

[15] CHOMSKY, N. - Formal properties of grammars. In:_____. Handbook of mathematical psychology. New York, John Wiley, 1963, v.2, p.323-418.