TRADUTOR MICROLOBAN

SERGIO HENRIQUE CRIVOROT

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS (M.Sc.)

Aprovada por:

ESTEVAM GILBERTO DE SIMONE - Presidente

JOSÉ LUCAS MOURÃO RANGEL NETTO

MICHAEL ANTHONY STANTON

BEATRIZ ZAKIMI MIYASATO

RIO DE JANEIRO, RJ - BRASIL

JUNHO DE 1983

CRIVOROT, SERGIO HENRIQUE

TRADUTOR MICROLOBAN (Rio de Janeiro, 1983)

VII, 117 p. 29,7 cm (COPPE-UFRJ, M.Sc. Sistemas, 1983).

Tese - Univ. Fed. Rio de Janeiro. Fac. Engenharia.

1. TRADUTOR MICROLOBAN. I.COPPE/UFRJ.II. Título (série)

A MEUS PAIS

AGRADECIMENTOS

- A meu orientador ESTEVAM que, não sei como, me convenceu a aceitar este trabalho. Agradeço por tudo que aprendí e pela sua atenção constante, mesmo após (infelizmente) ter-se desligado da COPPE.

- À ESSO BRASILEIRA DE PETRÓLEO S.A., que me permitiu cursar e concluir o Mestrado.

- À EXXON QUÍMICA S.A., em cujo computador o Tradutor MICROLOBAN foi desenvolvido.

- A meus amigos, pelo estímulo; especialmente à MARIA TERESA, pelo apoio e carinho.

- À RITA, minha artista gráfica favorita.

RESUMO

Microloban é um subconjunto da linguagem de operação de Banco de Dados Loban, que é suportado pelo Sistema de Gerência de Base de Dados Microban.

Este trabalho descreve o projeto e a implementação daqueles módulos da interface Loban que são do interesse da área de processadores de linguagens.

São descritos:

- o analisador léxico;
- o analisador sintático descendente, determinístico, com um símbolo de avanço construído a partir de uma gramática com lados direitos regulares através do método RRP LL(1). {5}
- um novo tradutor dirigido por sintaxe que utiliza uma gramática de saída gerando árvores binárias e que denominamos dendro-tradutor RRP.

São descritos ainda a maneira de usar esta implementação e o desenvolvimento de um programa que desenha as árvores geradas.

ABSTRACT

Microloban is a subset of the database operation language Loban, provided by the Microban Database Management System.

This thesis describes the design and development of the language processing modules:

- A common lexical analyser;
- A top-down deterministic parser with one symbol of look-ahead, generated from a regular right part grammar using RRP LL(1) method. {5}
- A specially developed syntax-directed translator with an output grammar producing binary trees - named RRP dendrum translator.

We describe how to use this language processor and a graphical output program which prints the translated trees.

ÍNDICE

I. INTRODUÇÃO

Microban é um sistema de gerência de base de dados interativo ou para processamento em lotes, autocontido e monousuário, e que suporta um subconjunto da linguagem de operação de banco de dados Loban{1}. Este subconjunto, denominado Microloban, é composto de comandos de definição, gerência e manipulação de base de dados, além dos de entrada e saída dos dados.

A Figura I.1, na página 2, mostra a arquitetura geral do sistema.

Este trabalho descreve a implementação do tradutor e seus módulos componentes. O objetivo principal foi discutir as idéias por trás da implementação, os caminhos que levaram à forma resultante, os aspectos teóricos envolvidos em cada módulo. Evitou-se entrar demasiadamente em detalhes do programa, muito mais adequados em um relatório técnico do que possivelmente ficariam em uma tese de mestrado.

O Capítulo II traz uma breve história do projeto Miniban, desde o seu início até o envolvimento do autor deste trabalho. Pretende-se esclarecer o leitor a respeito do significado desta implementação com relação ao projeto como um todo.

O Capítulo III descreve sucintamente as principais características da linguagem de operação de banco de dados, Loban, e da interface Microloban.

Na primeira parte do capítulo são definidos os conceitos introduzidos pela linguagem, isto é, os conceitos com os quais um usuário do sistema de gerência de banco de dados terá de se familiarizar.

Na segunda parte são listados alguns tipos pre-definidos em Microloban, comandos da linguagem, e algumas funções.

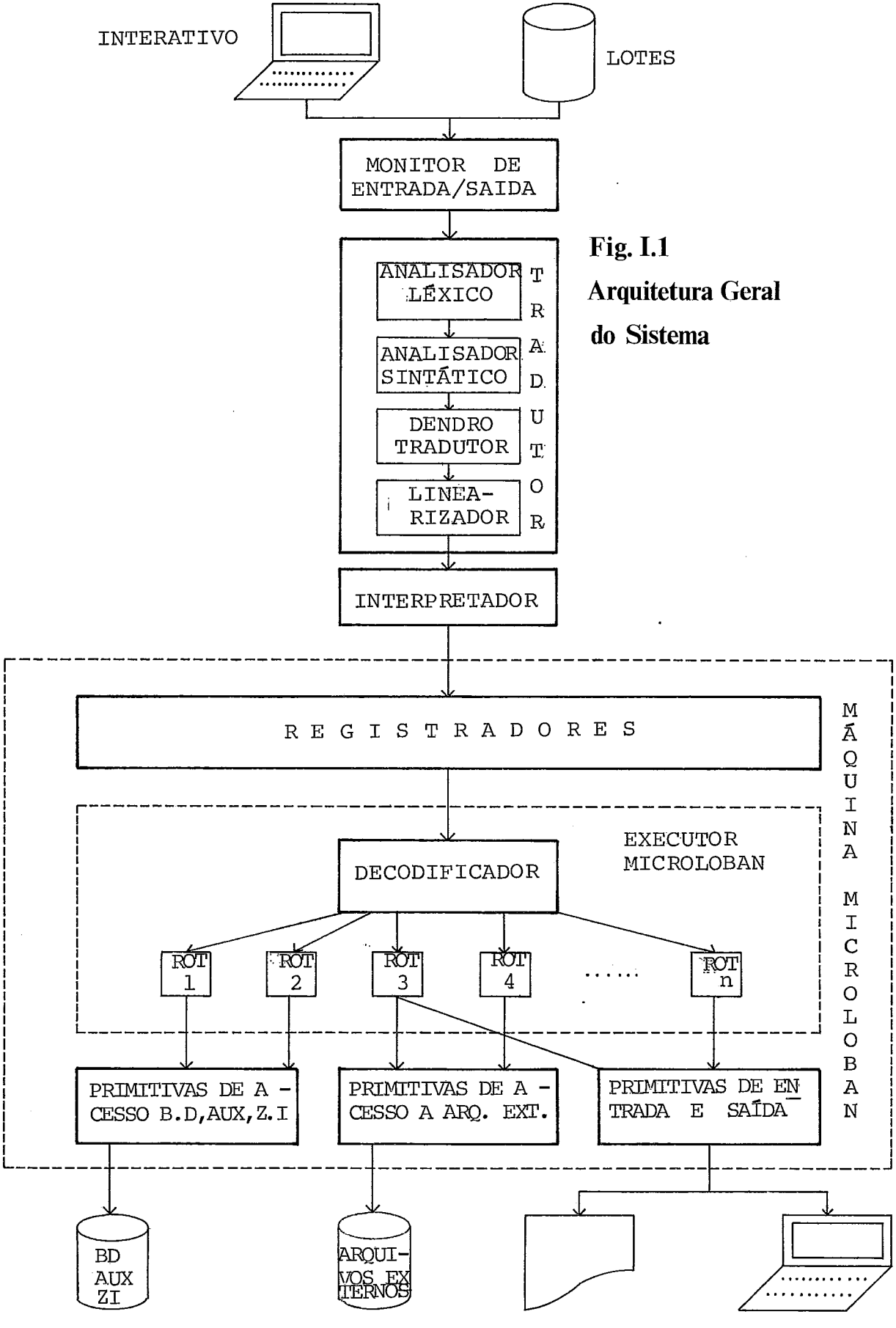


Fig. I.1
Arquitetura Geral do Sistema

No Capítulo IV é descrito o analisador léxico, modelado através de um transdutor finito, que traduz o programa fonte para uma sequência de unidades sintáticas que servirá de entrada para o analisador sintático.

O Capítulo V forma, juntamente com o Capítulo VI, o coração do trabalho.

No Capítulo V é estabelecida a base teórica necessária para a descrição do dendro-tradutor RRP. Os conceitos vão sendo introduzidos um a um, e no desenrolar do Capítulo vão se interrelacionando.

As gramáticas LL (1) são apresentadas. Apresenta-se então as gramáticas RRP, até se chegar às RRP LL(1), para as quais se constroi um analisador sintático. A seguir define-se árvores binárias e suas possíveis representações. Depois, o esquema de tradução dirigido pela sintaxe mostra a maneira de se realizar a geração de código simultaneamente com a análise sintática. Finalmente, todas as idéias são fundidas para o modelo de um SDTS, com gramáticas de entrada RRP LL(1), que emite árvores binárias na saída.

No Capítulo VI a teoria é colocada em prática nas seguintes etapas: a representação do Microloban através de uma gramática RRP-LL(1) e a implementação de seu analisador sintático; a definição do código intermediário a ser gerado para o interpretador Microloban, em forma de árvore binária linearizada; e a implementação do SDTS que o gera. Aborda-se ainda a estratégia de recuperação de erros utilizada.

O Capítulo VII mostra exemplos da utilização do tradutor, inclusive para fins didáticos, conjugado com o programa que desenha árvores, que permite uma representação gráfica do código intermediário gerado pelo tradutor Microloban.

Os demais capítulos são auto-explicativos.

II. HISTÓRICO

O Projeto Miniban - projeto de um sistema de banco de dados em minicomputador Nacional - começou a ser desenvolvido no ano de 1977 como resultado de um convênio entre o CNPQ, GMD da República Federal da Alemanha, Digibras e a UFRGS.

Miniban tem como objetivo a especificação de um sistema de banco de dados para um Minicomputador nacional, assim como o desenvolvimento de tecnologia nacional na área de banco de dados. A primeira etapa do projeto Miniban concentrou-se na especificação das estruturas de informação, as operações usando estas estruturas, assim como a definição da linguagem de operação de banco de dados Loban. A primeira implementação de um subconjunto Loban, denominado Sistema L, está sendo desenvolvido pela UFRGS. A partir de outubro de 1978 a COPPE/UFRJ iniciou sua participação neste projeto, dando origem ao projeto Miniban/COPPE.

A fase inicial do projeto Miniban/COPPE consistiu da definição detalhada de Loban, com a principal característica de separar nitidamente a etapa de definição da interface usuário/banco de dados, da etapa de realização ou implementação da mesma em um sistema portador.

Devido à extensão e extrema complexidade da interface assim definida, e ao desejo de adaptar Loban a um computador de pequeno-porte, a equipe do projeto Miniban/COPPE optou, como segunda etapa do projeto (denominado Microban), pelo desenvolvimento de um protótipo capaz de suportar um subconjunto Loban.

O sistema portador escolhido foi o Cobra-300 por ser uma máquina nacional e pela necessidade de desenvolvimento de software para sistemas deste porte.

O subconjunto Loban definido para a implementação do primeiro protótipo é denominado Microloban.

A primeira etapa da implementação deste protótipo é o desenvolvimento dos módulos de análise léxica e sintática e do tradutor para código intermediário.

Estas etapas são tarefas a serem executadas por pessoal que tenha como principais áreas de interesse linguagens de programação e o desenvolvimento de compiladores; e como área secundária, banco de dados.

Este é o caso do autor, que não pertence à equipe do projeto Miniban/COPPE. O trabalho, nominalmente, foi iniciado em 1981 mas todo este ano e a primeira metade de 1982 foram gastos ainda na definição do subconjunto a ser implementado. Pouco foi efetivamente feito no desenvolvimento do analisador e tradutor. O esforço maior foi, portanto, desenvolvido durante o segundo semestre de 1982 e o primeiro semestre de 1983.

Provavelmente, do ponto de vista de banco de dados, este trabalho padece de alguns males causados pelo seu desenvolvimento quase que totalmente em separado da equipe do projeto. Não é este o método que o autor recomenda a ninguém.

Como se trata de um conjunto de módulos e não da implementação total do Microloban, o autor teve o cuidado de escrever um programa de desenho de árvores que permite a fácil leitura da saída do tradutor, para auxiliar no desenvolvimento de etapas posteriores e, eventualmente, no ensino de Loban.

Espera-se que o tradutor venha efetivamente a ser utilizado por se tratar de uma pequena parte do grande esforço que deve ser desenvolvido para a efetiva geração de tecnologia nacional.

III. A LINGUAGEM LOBAN

Este capítulo tem como base a referência {2}, da qual o autor é um dos co-autores. Para um estudo detalhado do Loban, no entanto, recomenda-se {1}.

1. CARACTERÍSTICAS PRINCIPAIS DE LOBAN

A linguagem de operação de banco de dados (Loban) é uma linguagem autocontida que engloba comandos de definição, manipulação e gerência de dados, controle de acesso, providências de reconstrução, tratamento de erros e definição de transações, além de facilidades oferecidas pelas linguagens convencionais como definição de macros, comandos iterativos, condicionais e de entrada e saída de dados.

O desenvolvimento de Loban está baseado em conceitos IMC ("Information Management Concepts"), cuja principal característica é a distinção entre (construção de) informação (comumente denominado "valor" ou "conteúdo" de uma variável) e sua ocorrência dentro de um determinado contexto (conhecido como "nome de variável" ou "endereço de uma variável").

O conceito de construção de informação ou simplesmente construção, define qualquer "pedaço de informação" que possa ser referenciado por uma interface de gerência de base de dados. Uma construção é considerada elementar (ítem) se ela não é composta de outras construções, em oposição a um agregado que é formado por várias construções chamadas "componentes imediatos". Caso os componentes imediatos de um agregado possuam nomes, este é denominado nominação; caso contrário, ele é uma coleção.

Na abordagem relacional, a estrutura básica é a nominação de ítens - TUPLA. Um coletivo de Tuplas (com o mesmo conjunto de nomes) forma uma tabela relacional (conhecida na literatura como relação).

Na abordagem em redes é formada uma unidade de informação, ou seja, uma construção ligando um registro chamado "owner" a um conjunto de registros chamados "members". Em Loban este tipo de construção é denominado ligação, que é uma nominação de 2 elementos: uma tupla sob nome L e uma tabela relacional sob nome T. Um coletivo de ligações forma uma tabela ligacional ("set" em Codasyl).

Um arquivo em Loban é uma nominação sobre uma tabela e uma construção sob nome ficha, que contém informações como data de criação, atualização, etc. Dependendo do tipo de tabela componente, o arquivo será relacional ou ligacional.

Opcionalmente, Loban permite definir uma ou mais ordenações sobre a tabela de um arquivo. O agrupamento de arquivos constitui o acervo de trabalho (ACTRAB) que, juntamente com as construções usadas para providências de reconstrução, formam o acervo setorial (ACSET) que será o ambiente de trabalho para uma determinada aplicação. Cada acervo setorial tem associado a ele um nome, e o conjunto de todos os acervos setoriais forma o acervo total (base de dados), que é a construção mais abrangente suportada por Loban.

Para referenciar construções, Loban utiliza expressões denominadas endereços de pontos, que localizam construções dentro de um determinado contexto. Em geral, um endereço de ponto é uma seqüência de expressões booleanas, separadas pelo caracter ponto (.), que avaliadas num determinado nível de um agregado determinam os componentes imediatos que são selecionados (referenciados). Para que um componente imediato de um agregado seja selecionado, a expressão booleana correspondente deverá resultar no valor "verdadeiro".

Um recurso importante suportado por Loban é a marcação de pontos, isto é, depois de ter endereçado pontos na base de dados o usuário pode marcar estes pontos, atribuindo-lhes um nome, definindo "views" e "snapshots", cuja principal utilidade é

permitir referenciar estes pontos (em outros lugares) através do nome da marca, sem a necessidade de reavaliar o endereço de ponto que os selecionou.

O termo pretipo, em Loban, é usado para alguns tipos de construções previamente definidos na linguagem, como por exemplo: real, inteiro, tupla, tabela, etc. Todo usuário tem a possibilidade de definir os seus "próprios" tipos de construções. Toda definição de um tipo de construção é feita através de verbetes de coerência, que, em geral, constituem a definição das regras de consistência da base de dados.

Além dos verbetes de coerência, existem em Loban verbetes de usuário (para identificar os usuários do sistema), verbetes de acesso (para regulamentar os acessos a base de dados pelos diferentes usuários), entre outros. Todos estes verbetes são agrupados numa construção denominada folha (semelhante ao "schema" da Codasyl).

Todo usuário Loban possui uma área de trabalho denominada canal auxiliar, que funciona como uma base de dados particular e temporária, na qual ele pode armazenar, manipular e referenciar construções livremente.

Em geral, todos os resultados intermediários das operações são obtidos na zona intermediária, cujo conteúdo é perdido após a execução completa de um comando.

2. CARACTERÍSTICAS DA INTERFACE MICROLOBAN

Microloban é uma interface cuja especificação foi realizada sem a preocupação de como suas estruturas seriam representadas internamente na máquina escolhida para implementação. Esta separação deu maior liberdade de implementação, pois pode-se escolher dentre um maior número de alternativas, qual a solução mais viável quanto a realização no sistema portador.

Suas estruturas de informação fazem com que se tenha uma visão global da informação, e as operações sobre estas resolvem a maioria dos problemas de tratamento da informação na área de banco de dados. Tentou-se dar na sua definição a possibilidade de futuras extensões sem modificar suas funções originais.

Por ser uma interface poderosa, a proposta inicial é que sua definição e implementação se preste como uma ferramenta de ensino na área de banco de dados, e que funcione como um sistema de referência na comparação das diversas abordagens.

Microloban é uma linguagem autocontida, em português, que engloba diversas funções, dentre as quais estão as que descrevem a informação e seu relacionamento e as estruturas da base de dados (DDL), as funções de manipulação e uso da base de dados (DML), e funções normalmente realizadas por utilitários do SGBD, como reconstrução de acervos, modificações das definições dos dados, etc.

O principal critério adaptado na definição do subconjunto que compõe microloban foi o de reduzir a complexidade de Loban a um nível aceitável no sistema portador, porém mantendo a sua filosofia básica.

A comunicação do usuário com o sistema de banco de dados é feita utilizando uma instrução de trabalho, a qual corresponde a realização de um serviço. Uma instrução de trabalho é constituída de uma instrução de início, uma lista de instruções autôno -

mas e uma instrução de fim. Esta instrução de início determina se o processamento será interativo ou em lotes. Os dados a serem processados serão fornecidos ou junto com as instruções como anexos, ou separados em arquivos externos. Como resposta são fornecidos o resultado externo (ex.: relatórios), e o interno - que é guardado como uma base de dados caso tenham sido feitas alterações sobre a mesma. Serão também fornecidas mensagens operacionais padronizadas informando as ocorrências durante o processamento.

No que diz respeito a entrada/saída de dados, será utilizada uma única máscara (formato) padrão predefinida e várias regras de interpretação e representação, respectivamente.

Microloban permite alguns dos pretipos de Loban, entre os quais podemos citar:

- Pretipos atômicos: real, inteiro;

- Pretipos agregados básicos: numeração de caracteres, data, hora, tupla, ligação, coleção de ítens, tabelas relacionais e ligacionais;

- Folha, que conterá:
  - Descrição dos tipos de construção que compõem a base de dados (verbetes de coerência)
  - Autorizações permitidas (verbetes de acesso)
  - Identificação de usuários (verbete de usuário)
  - Procedimentos pre-definidos (verbete de texto fonte)

- Arquivo, composto de um tipo de construção padrão chamado ficha e uma tabela. Sendo assim, existem arquivos relacionais e ligacionais. Uma restrição feita em Microloban é a eliminação das relações de ordem tanto dos arquivos quanto das ligações.

- Ficha, contendo informações referentes ao arquivo ou acervo - setorial tais como: data de criação, data da última atualização, etc.

- ACTRAB (acervo de trabalho)

- ACSET (acervo setorial)

- ACTOT (acervo total)

Microloban permite os seguintes comandos:

- Comandos de gerência, onde o usuário tem a possibilidade de criar e abolir tanto acervos setoriais quanto arquivos.

- Comandos de manipulação, que permitem incluir, excluir e substituir construções tanto na base de dados quanto no canal auxiliar.

- Comandos de controle de fluxo, que permitem a execução repetitiva e/ou condicional . O comando iterativo permite a especificação de uma ordenação temporária sobre o conjunto de pontos a serem processados.

- Comandos de alocação de recursos com os quais o usuário informa a base de dados a ser usada, assim como os dispositivos de entrada/saída necessários. Além de definir qual a base de dados requisitada, o usuário deverá especificar a área de dados a ser usada (protegida) e para que tipo de acesso.

- Comandos de marcação, que permitem a definição de "snapshots", e não de vistas como é o caso de Loban.

- Comandos de reconstrução. São permitidos dois níveis de reconstrução: de sessão (de tipo regressiva) e de comandos dentro de uma sessão (tanto progressiva quanto regressiva).

- Comando de saída que permite representar construções no meio externo.

- Comando de definição de transação, que permite representar - conjunto de comandos cuja execução será considerada como uma unidade de processamento. No início da transação são es - pecificadas as ações que serão executadas quando da ocorrên - cia de êrros de execução.

Além dos comandos acima descritos temos as expressões que quando executadas geram construções na zona intermediária (área de trabalho). Estas expressões não têm restrições quanto ao nível de embutimento e englobam as seguintes funções:

- Entrada de dados
- Operações aritméticas
- Operações da álgebra relacional
- Operações do cálculo relacional
- Operações sobre tabelas relacionais e ligacionais

Microloban não permite operações sobre cadeias de caracteres, - produto e concatenação cartesiana.

Por último, temos as expressões que permitem o endereçamento de pontos do acervo ou canal auxiliar, e campos dos volumes de entrada e saída.

IV. ANÁLISE LÉXICA

O desenho de um analisador léxico eficiente é tarefa simples. Farta teoria a respeito pode ser encontrada em {6} e {8}, que são as referências para todo o capítulo. A notação usada é a de {6}.

A função do analisador léxico é grupar seqüências de caracteres terminais em entidades sintáticas primitivas, conhecidas como - "Tokens". O que vai constituir ou não uma entidade sintática em uma dada implementação, embora se trate de uma decisão do projetista, é fundamentalmente influenciado pela especificação da linguagem.

A cada seqüência de símbolos terminais grupados, associa-se uma estrutura léxica consistindo de um par da forma (tipo, valor).O primeiro componente é um tipo de entidade sintática, tal como "identificador", e o segundo componente fornece informações que individualizam um elemento dentro do conjunto de seqüências de símbolos que pertencem ao mesmo tipo.

O primeiro componente do par é usado pelo analisador sintático, enquanto que o segundo é usado durante a fase de geração de código.

Assim, o analisador léxico é um tradutor cuja entrada é uma seqüência de símbolos representando o programa fonte e cuja saída é uma seqüência de entidades sintáticas primitivas. Esta saída forma a entrada do analisador sintático.

A melhor representação para as entidades sintáticas reconhecidas pelo analisador léxico é em forma de expressão regular.

Uma expressão regular (ER) e o conjunto de seqüências sobre o alfabeto Σ que denota são definidas como:

(1) $\emptyset$ é uma ER e denota o conjunto vazio;
(2) a pertencente a Σ é uma ER e denota o conjunto $\{a\}$;
(3) Se p e q são ER's denotando os conjuntos P e Q, então:
  (p|q) é ER e denota $P \cup Q$;
  (pq) é ER e denota PQ ;
  (p)* é ER e denota P*;
(4) Nada mais é ER.

Os parênteses redundantes podem ser removidos obedecida a precedência: fechamento, alternação, concatenação.

São aceitas também abreviaturas, com a mesma precedência que o fechamento:
(1) p? denotando $P \cup \{\varepsilon\}$;
(2) p+ denotando PP*;
(3) p£q denotando P(QP) *.

Na página 15, a figura IV.1 mostra algumas das entidades sintáticas reconhecidas em Microloban. São mostrados: a representação externa em forma de expressão regular e o par associado (tipo de entidade, valor). A tabela de tradução completa com todas as entidades sintáticas está no apêndice 1.

A representação externa está em forma de expressão regular sobre o alfabeto formado por todos os caracteres válidos no Cobra 300:

Σ ={"A", "B", "C", ..."Z", "0", "1", ..., "9", ";", ":", ...}

Algumas simplificações foram adotadas:
l = "A" | "B" | "C" | ... | "Z"
d = "0" | "1" | "2" | ... | "9"

As outras entidades com a mesma lei de formação que os identificadores são as palavras reservadas do Microloban, isto é, identificadores com significado pre-estabelecido.

FIGURA IV.1 - Parte da tabela de tradução do analisador léxico

REPRESENTAÇÃO	TIPO	VALOR
";"	PONTO-E-VÍRGULA	
","	VÍRGULA	
"."	PONTO	
"<"	MENOR	
">"	MAIOR	
":"	DOIS-PONTOS	
")"	FECHA-PARENTESES	
"("	ABRE-PARENTESES	
"<>"	DIFERENTE	
"<="	MENOR-OU-IGUAL	
">="	MAIOR-OU-IGUAL	
":="	ATRIBUIÇÃO	
$l(l/d/"\_")^*$	IDENTIFICADOR	REPRESENT.EXTERNA
d+	INTEIRO	REPRESENT.EXTERNA
d+ "," d+	REAL	REPRESENT.EXTERNA
dd "." dd "." $(dd)^?$ dd	DATA	REPRESENT.EXTERNA
dd ":" dd $(":" dd)^?$	HORA	REPRESENT.EXTERNA
ASPAS Σ^* ASPAS	NUMCAR	REPRESENT.EXTERNA
"A"	A	
"C"	C	
"AO"	AO	
"CC"	CC	
"PC"	PC	
"COM"	COM	
"REC"	REC	
"SOBRE"	SOBRE	
"ALTERAR"	ALTERAR	
"EXCLUIR"	EXCLUIR	
"COLITENS"	COLITENS	
"COMPILAR"	COMPILAR	
"GERENCIAR"	GERENCIAR	
"REPRESENTAR"	REPRESENTAR	
"ALFANUMÉRICO"	ALFANUMÉRICO	
"TELEIMPRESSORA"	TELEIMPRESSORA	

A decisão do que é um token depende não só da linguagem, mas também do projetista do analisador. Por exemplo, poder-se-ia ter optado por não considerar a existência de um token do tipo literal data, mas sim considerar uma seqüência de cinco tokens: inteiro - ponto-inteiro-ponto-inteiro. Simplificações como esta levariam a um analisador léxico mais simples, mas ao custo de transferir o reconhecimento de um literal-data para a fase de análise sintática. Este enfoque, no entanto, é bastante discutível. Foi seguida a orientação segundo a qual é melhor complicar-se o analisador léxico, para simplificar o analisador sintático. A lista de "tokens", mostrada acima, é aquela que julgou-se estar mais próxima da estrutura do Loban, como definida pelo usuário.

Sempre que o analisador sintático precisa de uma nova entidade sintática, o analisador léxico é chamado, sendo, portanto, uma subrotina do analisador sintático. O analisador léxico só retornará o controle ao analisador sintático (com exceção do caso de fim de arquivo) após reconhecer uma entidade sintática válida para seu uso. Caso contrário continuará analisando o registro lido, enquanto for necessário. O analisador léxico escolherá sempre a mais longa entidade sintática possível.

Duas definições são importantes para a concepção do analisador léxico: automato finito determinístico e transdutor finito.

Automato finito não determinístico é uma 5-Tupla

$$M = (Q, \Sigma, \delta, q_0, F)$$

onde:

(1) Q é um conjunto finito de estados;

(2) Σ é um conjunto finito de símbolos;

(3) δ, a função de transição,

$$\delta: (Q \times \Sigma) \rightarrow Q^*$$

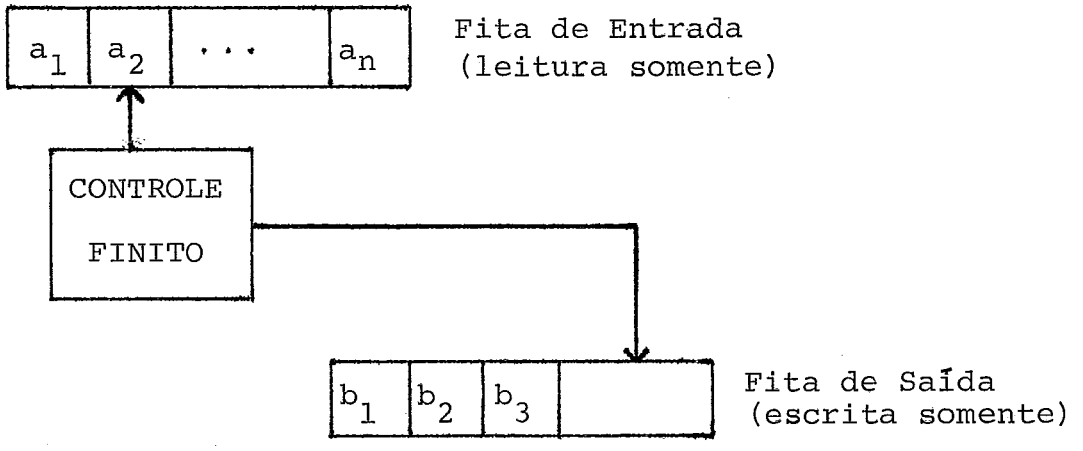
(4) q_0 é o estado inicial;

(5) F, que é um subconjunto de Q, é o conjunto de estados finais.

Se δ mapeia $(Q x \Sigma)$ em Q diremos que o autômato é determinístico - (AFD).

Como já foi exposto, o analisador léxico é um tradutor, que traduz o programa fonte para uma sequência de entidades sintá - ticas.

O tradutor mais simples é o transdutor finito, usado por nós. Um transdutor é composto por um reconhecedor acoplado a um trans - missor, que emite um conjunto de símbolos de saída a cada movi- mento feito, podendo este conjunto ser vazio. O transdutor fi - nito é obtido tomando-se um autômato finito e permitindo a má - quina emitir um conjunto de símbolos de saída em cada movimen - to. A Figura IV.2 apresenta o modelo de um transdutor finito.



Um transdutor finito é uma 6-Tupla $(Q, \Sigma, \Delta, \delta, q_0, F)$ onde,

(1) Q é um conjunto finito de estados

(2) Σ é um alfabeto de entrada

(3) Δ é um alfabeto de saída

(4) δ é uma função de transição

$$\delta: (Q \times \Sigma) \rightarrow \mathcal{P}(Q \times \Delta^*)$$

(5) q_0 pertence a Q, sendo o estado inicial

(6) F está contido em Q, sendo o conjunto de estados finais

A análise léxica é dita direta quando, dada uma seqüência de símbolos de entrada e um ponteiro para aquela seqüência, o analisador determina a qual tipo de entidade sintática pertence o grupo de símbolos imediatamente a direita do lugar apontado e move o ponteiro para a direita daquele grupo de símbolos.

A maneira mais eficiente de se implementar um analisador léxico direto é através da busca em paralelo, pois a cada símbolo lido o número de possibilidades decresce rapidamente.

A estratégia de desenho adotada foi a seguinte:

Para cada tipo de entidade sintática foi desenhado um automato finito que a reconhecesse. Todos os automatos foram então combinados em um só, que foi tornado determinístico e mínimo.

A partir do automato finito determinístico combinado, obteve-se um transdutor finito simples que emite na saída o tipo de entidade sintática reconhecida e, eventualmente, alguma informação particular sobre seu valor.

Cada estado do automato representa estados de vários dos automatos componentes. Quando o automato combinado entra num estado que contem um estado final de um dos automatos componentes, e nenhum outro estado, ele para e emite o nome da entidade sintática reconhecida, se não houver novas transições possíveis.

O automato finito determinístico combinado, representado pelo analisador léxico está na Figura IV.3, na página 19.

Um problema ocorre no caso dos identificadores e das palavras reservadas. Devido a grande quantidade de identificadores possíveis, usa-se uma definição simplificada (letra seguida de qualquer combinação de letras, digitos e travessões) para o transdutor.

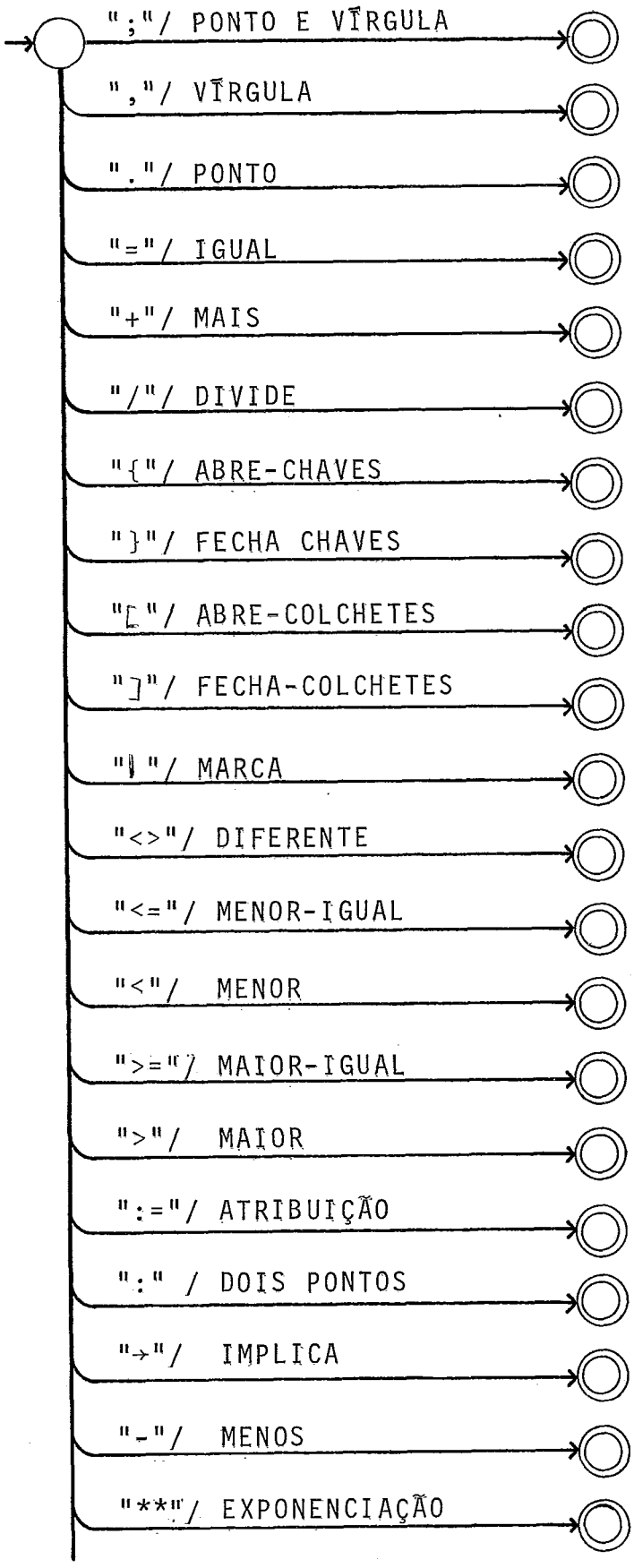
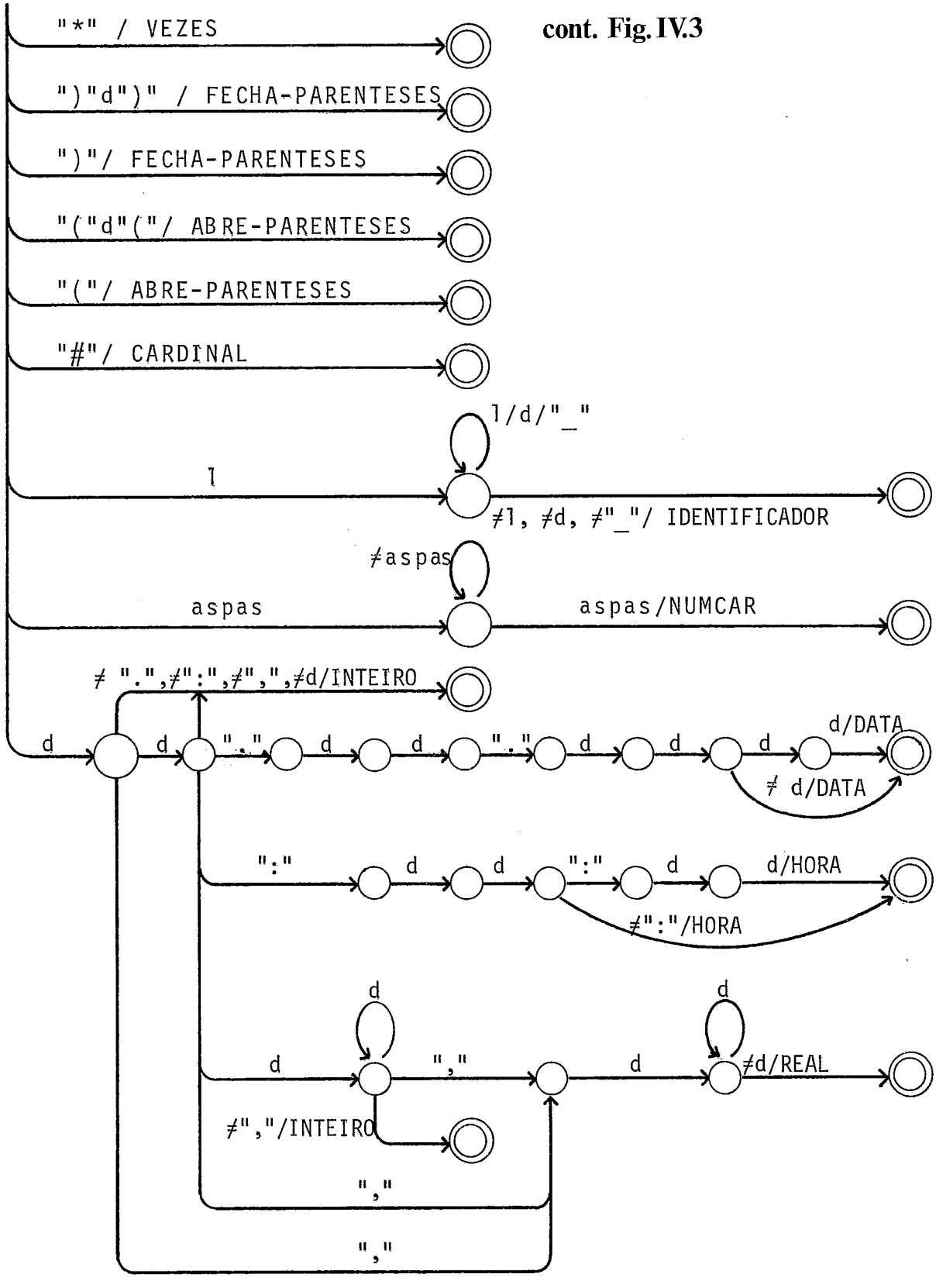


Fig. IV.3

Autômato Finito


"*" / VEZES
")"d")" / FECHA-PARENTESES
")"/ FECHA-PARENTESES
"("d"("/ ABRE-PARENTESES
"("/ ABRE-PARENTESES
"#"/ CARDINAL
l/d/"_"
l
≠l, ≠d, ≠"_"/ IDENTIFICADOR
≠aspas
aspas
aspas/NUMCAR
≠ ".",≠":",≠",",≠d/INTEIRO
d
d
"."
d
d
"."
d
d
d
d/DATA
≠ d/DATA
":"
d
d
":"
d
d/HORA
≠":"/HORA
d
d
d
","
d
≠d/REAL
≠","/INTEIRO
","
","

cont. Fig. IV.3

Quando o transdutor reconhece um identificador, é executado um procedimento especial para determinar se ele pertence ao conjunto pre-definido, representado sob a forma de uma tabela. Caso isto ocorra, trata-se de uma palavra reservada, e não de um identificador.

A tabela de palavras reservadas é representada na memória como uma cadeia contínua de caracteres a partir da posição denominada "Tabreserva", ocupando 1130 bytes. As palavras reservadas lá são colocadas em ordem ascendente de comprimento e em ordem alfabética. Deste modo, temos todas as palavras de comprimento 1, depois todas as de comprimento 2, etc. A última palavra de cada grupo começa com um câracter invâlido em palavras, o caracter "}", para indicar o final do grupo de palavras daquele comprimento.

Abaixo vai descrito o algorítmo de busca na tabela de palavras reservadas:

Passo 1: Inicial := função (comprim do ident, letra inicial)

Passo 2: Vezes := 0

Passo 3: Posição := tabreserva + inicial
+ (vezes * comprim do ident)

Passo 4: Compara caracter a caracter o identificador sendo analisado com o conteúdo das posições de memória {Posição:Posição + Comprim - 1}

Passo 5: Se há igualdade, gera o token correspondente a palavra reservada:
Token:= função (posição, comprim do ident)

Passo 6: Se não há igualdade e a letra inicial do identificador sendo buscado é menor ou igual a letra em{posição}, então incrementa "vezes" e vai para o passo 3.

Passo 7: Se não há igualdade e a letra inicial do identificador sendo buscado é maior que a letra em{posição}, então o identificador em questão não é uma palavra reservada.

O algoritmo descrito acima foi elaborado visando-se um bom aproveitamento na memória, aspecto mais crítico desta implementação. Cada palavra utiliza apenas o seu comprimento real, sendo perdido apenas o espaço ocupado pela palavra fictícia colocada no final de cada grupo. Deste modo, 963 dos 1110 caracteres da tabela são efetivamente aproveitados, isto é, 87%.

As funções mencionadas nos Passos 1 e 5 são algebrismos bastante específicos, definidos a partir do conjunto de palavras reservadas no Loban, visando um bom aproveitamento da memória, como explicado acima. O aspecto lógico, que é fundamental, é explicado em detalhe no algorítmo.

Não há necessidade de manter-se uma tabela de identificadores. Toda a análise semantica estática será realizada pelo interpretador. O tradutor se limitará a passar para o interpretador uma descrição de cada identificador contido no programa lido, sem guardar nenhum registro a respeito.

V. ANÁLISE SINTÁTICA / GERAÇÃO DE CÓDIGO. DISCUSSÃO TEÓRICA

Neste Capítulo, com exceção da Seção 3, utiliza-se a notação e os conceitos básicos de teoria de conjuntos e teoria de linguagens desenvolvidos em {6}.

Na Seção 1 alguns conceitos bastante básicos são apresentados, tais como o de grâmatica, somente para um melhor encadeamento de idéias, sem que se pretendesse cobrir todo o alcance de um curso sobre linguagens

O estudo sobre gramáticas com lados direitos regulares e suas representações, desenvolvido na Seção 2, teve como referências {3} e {5}.

Maiores detalhes sobre o assunto abordado na Seção 3, Árvores Binárias, podem ser encontrados em {11}, cuja notação foi utilizada.

1. GRAMÁTICAS LL(1)

Um alfabeto é qualquer conjunto de símbolos.

Define-se uma seqüência de símbolos sobre um alfabeto Σ da seguinte maneira:

(1) ε é uma seqüência sobre Σ.

(2) Se x é uma seqüência sobre Σ e a está em Σ, então xa é uma seqüência sobre Σ.

(3) Nada mais é seqüência.

Uma linguagem sobre um alfabeto Σ é sempre um subconjunto de Σ^*. Linguagens de programação tais como FORTRAN e LOBAN estão obviamente incluídas nesta definição.

Uma linguagem composta de um número finito de seqüências de símbolos pode ser representada através de uma lista de todas essas seqüências. Para uma linguagem composta de um número infinito de seqüências outro método de representação necessariamente deve ser procurado.

Há diversos métodos de representação que preenchem este requisito. Usaremos um método generativo, chamado gramática. Cada sentença da linguagem pode ser construída através de métodos bem definidos, usando as regras (produções) da gramática. Representação através de gramáticas simplifica a análise sintática e a tradução, por causa da estrutura transmitida às sentenças da linguagem pela gramática.

Uma gramática é um sistema formal para se definir uma linguagem, bem como um dispositivo para dar às sentenças de linguagem uma estrutura útil.

Uma gramática é uma quadrupla $G = (N, \Sigma, P, S)$, onde:

(1) N é um conjunto finito de símbolos não-terminais (algumas vezes chamados de variáveis ou categorias sintáticas).

(2) Σ é um conjunto finito de símbolos terminais, disjunto de N.

(3) P é subconjunto finito de $N \times (N \cup \Sigma)^*$
Um elemento (α, β) de P será escrito na forma $\alpha \rightarrow \beta$ e será chamado uma produção.

(4) S é um símbolo particular em N chamado símbolo inicial.

Uma gramática da forma descrita é dita livre de contexto se cada produção de P é da forma $A \rightarrow \alpha$, onde A pertence a N e α pertence a $(N \cup \Sigma)^*$. Outros tipos de gramáticas com diferentes definições para P existem, mas não são relevantes neste estudo.

Como mencionado anteriormente, a saída do analisador léxico é uma seqüência de pares (tipo de entidade sintática, valor). Esta seqüência forma a entrada do analisador sintático, que analisa somente os primeiros componentes dos pares - os tipos. A informação sobre cada um - segundo componente - é usada mais tarde, na geração de código. Portanto, os primeiros componentes de pares são os terminais da gramática.

A análise sintática ou "parsing" é o processo em que a seqüência de pares é examinada para determinar-se se ela obedece certas convenções estruturais explícitas na definição sintática da linguagem, isto é, se ela pertence à linguagem.

A partir de um conjunto de regras sintáticas é possível construir-se automaticamente analisadores sintáticos, ou "parsers" que garantirão que um programa fonte obedece à estrutura sintática definida por estas regras sintáticas.

Restringindo-se à classe de gramáticas em estudo, é possível construir-se analisadores sintáticos mais eficientes. Discutiremos o caso de algorítmos de análise sintática caracterizados pelo fato de que a seqüência de entrada é lida somente uma vez da esquerda para a direita e o processo de análise é completamente determinístico.

Na verdade, estamos restringindo a classe de gramáticas livres de contexto de modo que possamos construir um "parser" esquerdo determinístico para a gramática sendo considerada.

Diversos conceitos utilizados a seguir serão considerados já definidos, encontrando-se no estudo de teoria de linguagens realizado em {6}. São êles:

- Derivação (=>)
- Derivação esquerda ($\underset{lm}{=>}$)
- "Parse"
- "Parsing" preditivo
- Forma sentencial (α, β, γ, ...)

Para uma gramática livre de contexto $G = (N,\Sigma,P,S)$, define-se - FIRST $(\alpha) = \{x \mid \alpha \overset{*}{\underset{lm}{=>}} x\beta\}$, onde lm significa derivação esquerda. Isto é, FIRST (α) consiste de todos os símbolos terminais iniciais das seqüências de símbolos que podem ser derivadas a partir de α.

Diz-se que a gramática G é LL(1), se sempre que há duas derivações esquerdas:

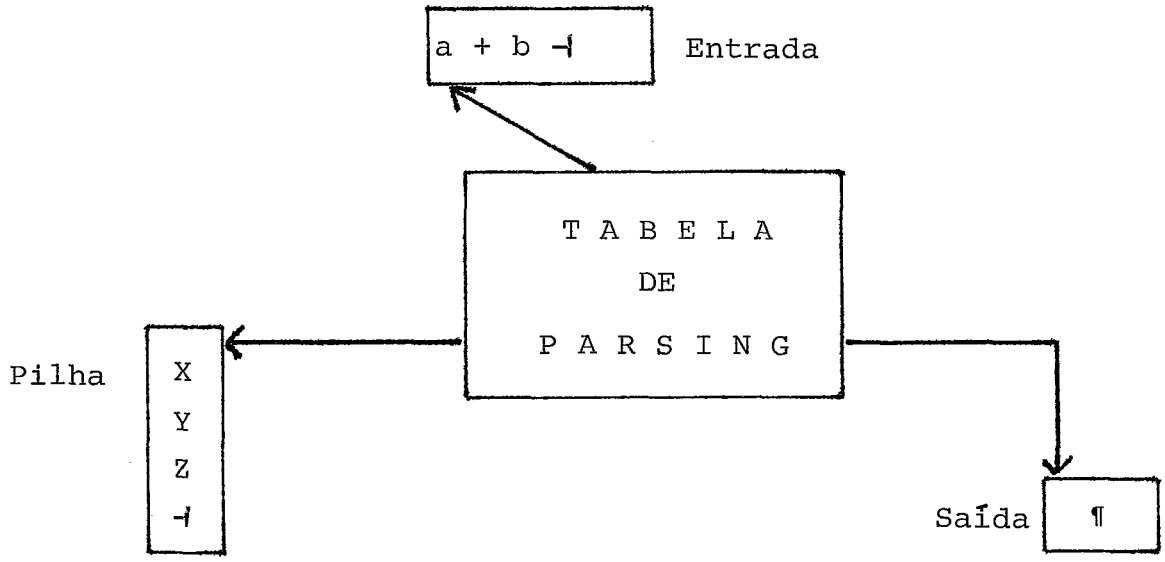
(1) $S \overset{*}{=>} wA\alpha \Rightarrow w\beta\alpha \overset{*}{\Rightarrow} wx$; e

(2) $S \overset{*}{=>} wA\alpha => w\gamma\alpha \overset{*}{=>} wy$,

tais que FIRST (x) = FIRST (y), então segue-se que $\beta = \gamma$.

Menos formalmente, G é LL(1) se dada uma seqüência de símbolos - $wA\alpha$ em $(N \cup \Sigma)^*$ e o primeiro símbolo terminal derivado de $A\alpha$, existe no máximo uma produção que pode ser aplicada a A para produzir uma derivação de qualquer seqüência de terminais começando - por w seguido por este terminal.

É possível efetuar-se o "parse" de gramáticas LL(1) de maneira - muito conveniente através de um algorítmo de parsing preditivo de um símbolo, usando-se uma fita de entrada, uma pilha e uma fita-de saída. O algorítmo preditivo de um símbolo tenta achar uma derivação esquerda da seqüência colocada em sua fita de entrada.



A fita de entrada contém a seqüência de entrada a ser analisada, seguida de ⊣ , o marcador de final. Ela é lida por uma cabeça capaz de ler o próximo símbolo, dito o símbolo de avanço.

A pilha contém uma seqüência de símbolos da gramática, precedidos de ⊣ . A tabela de controle do analisador é uma tabela bidimensional M(A,a), onde A é um não-terminal, e a é um terminal ou o símbolo ⊣ .

O analisador sintático é controlado por um programa que funciona da maneira descrita a seguir: o programa determina X, o símbolo no topo da pilha, e a, o símbolo atualmente na entrada. Estes dois símbolos determinam a ação do analisador. Há 3 possibilidades:

(1) Se X = a = ⊣, o analisador para e anuncia que a análise foi concluída com sucesso.

(2) Se X = a ≠ ⊣, o analisador desempilha X e avança o ponteiro da entrada para o próximo símbolo da entrada.

(3) Se X é um não-terminal, o programa consulta a entrada da tabela M(X,a). Esta entrada será ou uma produção da gramática, cujo lado esquerdo é o não-terminal X, ou será uma entrada de erro. Se M(X,a) = ($X \rightarrow UVW$), o analisador substitue no topo da pilha por WVU, com U no topo. Se M(X,a) = erro, uma rotina de recuperação de erros pode ser chamada.

(4) Em qualquer outra situação acusará erro.

Para descrevermos um algorítmo de construção da tabela de controle para análise sintática preditiva, falta-nos ainda uma ferramenta.

Seja $G=(N,\Sigma,P,S)$ uma gramática livre de contexto. Define-se FOLLOW (β), sendo β pertencente a $(N \cup \Sigma)^*$, como sendo o conjunto $\{w \mid S \overset{*}{\Rightarrow} \alpha\beta\gamma$ e w pertence a FIRST(γ)$\}$.

Isto é, FOLLOW (A) inclue o conjunto de símbolos terminais que podem ocorrer imediatamente a direita de A em qualquer forma sentencial.

A idéia por trás do algorítmo de construção da tabela é simples: seja $A \rightarrow \alpha$ uma produção com a pertencente a FIRST (α), então sempre que o analisador tem A no topo da pilha com a como símbolo na entrada, o analisador expande A por α. A única complicação ocorre quando $\alpha = \varepsilon$ ou $\alpha \overset{*}{\Rightarrow} \varepsilon$. Neste caso deve-se expandir também A por α se o símbolo na entrada pertence ao FOLLOW (A), ou se o $\dashv$ na entrada foi atingido e $\dashv$ pertence ao FOLLOW(A).

Dada uma gramática G, o algoritmo de construção da tabela de controle M do analisador sintático é o seguinte:

(1) Para cada produção $A \rightarrow \alpha$ da gramática, execute os passos (2) e (3).

(2) Para cada terminal a em FIRST (α), adicione $A \rightarrow \alpha$ na entrada M(A,a).

(3) Se ε pertence a FIRST (α), adicione $A \to \alpha$ na entrada $M(A,b)$ para cada terminal b pertencente a FOLLOW (A). Se ε pertence a FIRST(α) e $\dashv$ pertence a FOLLOW(A), adicione $A \to \alpha$ na entrada $M(A,\dashv)$.

(4) Marque cada entrada indefinida de M como erro.

2. GRAMÁTICAS RRP

A primeira definição relevante para o estudo de gramáticas RRP é a de expressão regular, mostrada na Seção 4.1.

Uma gramática com lados direitos regulares, dita RRP (do inglês "regular right parts"), livre de contexto, G é uma 4-tupla $G = (N,\Sigma,P,S)$ onde:

(1) N e Σ são conjuntos finitos de não terminais e terminais, respectivamente:

(2) P é um conjunto de pares $(A \to \pi)$ tal que A pertence a N e π é um conjunto regular sobre $(N \cup \Sigma)$;

(3) S é o símbolo inicial

Uma derivação em G é da forma: $uAv \Rightarrow uwv$ se $(A \to w)$ pertence a P, e w pertence a π.

A descrição de uma gramática RRP pode possuir vários formatos. Entre eles, são importantes para nós:

(1) Formato de expressão regular. Neste caso π é descrito por expressão regular sobre $(N \cup \Sigma)$.

(2) Formato de automato finito determinístico mínimo em que π é um AFD mínimo, o alfabeto de entrada é $(N \cup \Sigma)$ e δ mapeia $Q \times (N \cup \Sigma)$ em Q.

Para que uma gramática RRP seja LL(1) as condições são as seguintes:

Seja $G = (N,\Sigma,P,S)$ uma gramática RRP livre de contexto não recursiva à esquerda, ou seja:

- se A pertence a N então $A \not\Rightarrow A\alpha$

Ou ainda, se G estiver representada em formato de autômato finito:

- se $A_i \rightarrow M_i$ então Q_{0i} não pertence a F_i

G é LL(1) se:

(1) $\delta_i(q,X) = r$ e $\delta_i(q,B) = s \Rightarrow FIRST(X) \cap FIRST(B) = \{\}$
onde q,r,s pertencem a Q_i, B pertence a N, e X pertence a $(N \cup \Sigma)$;

(2) $\delta_i(q,X) = p$ e q pertence a $F_i \Rightarrow FIRST(X) \cap FOLLOW(A_i) = \{\}$
onde p e q pertencem a Q_i, X pertence a $(N \cup \Sigma)$.

Definiremos os conjuntos FIRST usando G e P como mostrados anteriormente da seguinte forma:

(A) FIRST (a) ={a} , com a pertencente a Σ;

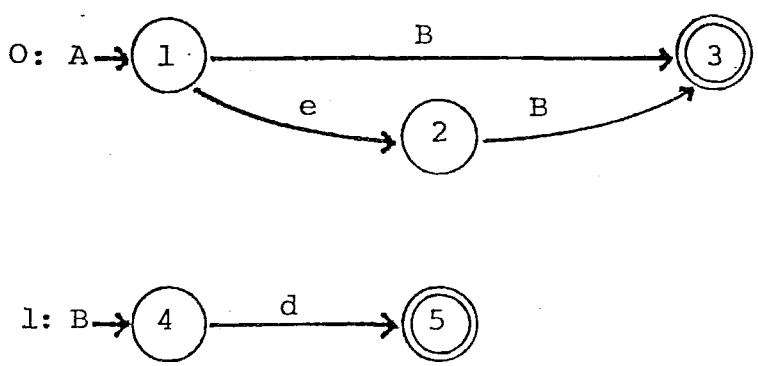
(B) Se $(A_i \rightarrow M_i)$ pertence a P então $FIRST(A_i) = \bigcup_j FIRST(X_j)$ com $\delta_i(Q_{0i}, X_j) = p$.

Note-se que se b pertence a FIRST(Y) então $Y \overset{*}{\Rightarrow} b\alpha$.

EXEMPLO:

$A \rightarrow (e)^{?}B$

$B \rightarrow d$



FIRST(A) = FIRST (B) U FIRST (e)

FIRST(A) = {d,e}

Com a mesma gramática G, o mesmo conjunto de produções P, podemos definir os conjuntos FOLLOW:

Seja $\delta_i(p,A_j) = r$, onde p e r pertencem a Q, e A_j pertence a N.

(A) Se r não pertence a F_i então FOLLOW $(A_j) = \bigcup_k$ FIRST(X_k), com X_K tais que $\delta_i(r, X_K) = s$;

(B) Se r pertence a F_i então FOLLOW $(A_j) = (\bigcup_k$ FIRST $(X_k)) \cup$ FOLLOW (A_i), para os mesmos X_k.

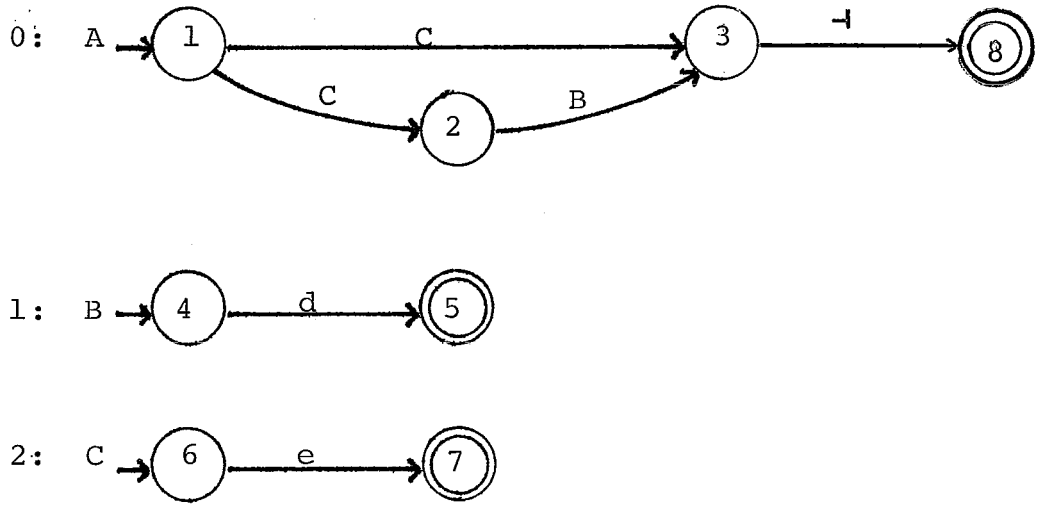
Note-se que se b pertence a FOLLOW (Y) então $S \overset{*}{\Rightarrow} \alpha Y b \gamma$.

EXEMPLO:

$A \rightarrow C(B)^{?}\dashv'$

$B \rightarrow d$

$C \rightarrow e$



FOLLOW (C) = FIRST (B) $\cup$ {⊣}

FOLLOW (C) = {d, ⊣}

Dispondo-se da gramática RRP escrita com os lados direitos representados por AFD mínimo, e os conjuntos FIRST e FOLLOW pode-se construir a tabela de controle para um analisador sintático.

A tabela de controle fornece uma ação a ser executada para cada par de $(Q \times (N \cup \Sigma \cup \{\dashv\})$, com os seguintes significados:

(A) (A,E) : Elimine o símbolo da entrada e faça o estado atual ser E. Corresponde a uma transição em um automato;

(B) (D,n) : Insira o estado atual na pilha sintática e faça o estado atual ser ENTRY(n) . Corresponde a uma derivação usando a produção n, emite o "parse" e inicia o

reconhecimento do automato n_i:

(C) (pop,S): coloque o não terminal S na entrada, como sendo o primeiro símbolo, e faça o estado atual receber o conteúdo do topo da pilha sintática, que de lá deve ser retirado;

(D) (Fim): indicando que a análise está concluída.

(E) (ERRO): todas as entradas indefinidas representam erros sintáticos.

O algorítmo de confecção da tabela de controle é muito simples. Recebendo-se um automato finito de múltipla entrada, que representa a gramática, executa-se os passos abaixo só preenchendo posições em TABCONTROLE se isso ainda não tiver sido feito. Caso contrário, para e avisa que a gramática não é LL(1):

(a) Para cada transição $\delta(p,X) = r$ faça TABCONTROLE(p,X) = (A,r). Além disso, se X pertence N faça TABCONTROLE (p,a) valer (D, X) para todo a pertencente a FIRST(X);

(b) Para todo f que seja estado final de uma produção $(A_i \rightarrow M_i)$ faça TABCONTROLE(f,b) = (pop,A_i), para todo b pertencente a FOLLOW (A_i);

(c) Para todo δ (p,x) =q, q pertencente a Fo, faça TABCONTROLE (p,x) = (FIM);

(d) Faça TABCONTROLE (p,x) = (ERRO) para toda entrada de TABCONTROLE ainda não preenchida.

A tabela de controle do analisador sintático, tal como foi definida se apresenta na forma de uma matriz de incidência.

Uma estrutura alternativa utilizaria listas, diminuindo o espaço ocupado, uma vez que a matriz é esparsa.

Em linguagem corrente, as condições para que uma gramática RRP - seja LL(1) são as seguintes:

(a) Em um dado estado, se for possível haver transição com dois-símbolos diferentes, a interseção dos FIRST'S dos dois símbolos tem que ser vazia.

(b) Se for possível haver transição com um determinado símbolo a partir do estado final de uma produção, a interseção do FIRST deste símbolo com o FOLLOW do não terminal que é o lado esquerdo da referida produção tem que ser vazia.

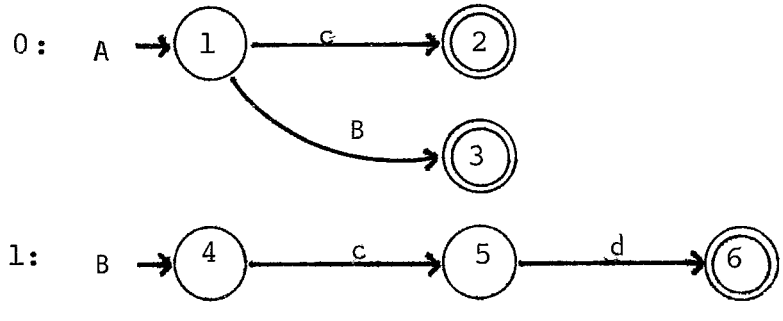
A primeira condição é necessária nos casos em que há transição a partir de um mesmo estado com um não terminal e um terminal. Se o terminal pudesse estar presente no FIRST do não terminal, o analisador não poderia decidir inequivocamente com base no próximo símbolo lido na entrada qual a ação que deve ser tomada, não prevalecendo a condição LL(1). A mesma restrição deve ser observada se houver transição a partir de um estado com dois não terminais.

A segunda condição é necessária para que o analisador possa decidir, quando estiver num possível estado final, se ele deve continuar na mesma produção. Se o símbolo for uma continuação válida, ou se realmente atingiu um estado final, caso o símbolo não seja uma continuação válida.

EXEMPLOS:

(1) A → c | B

B → cd

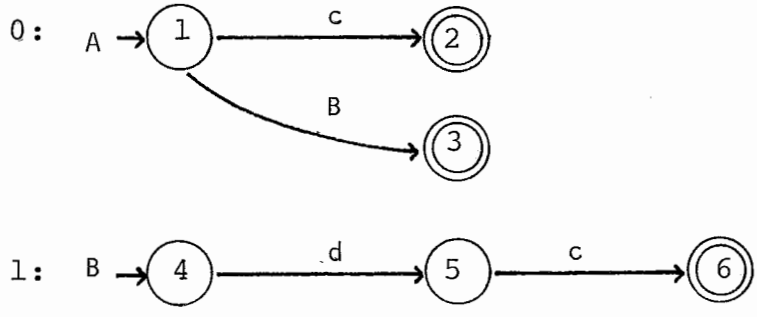


Esta gramática não é LL(1) pois:

1 - Existe transição com c e B no mesmo estado

2 - c está contido em FIRST (B)

(2) $A \rightarrow c|B$
$B \rightarrow dc$

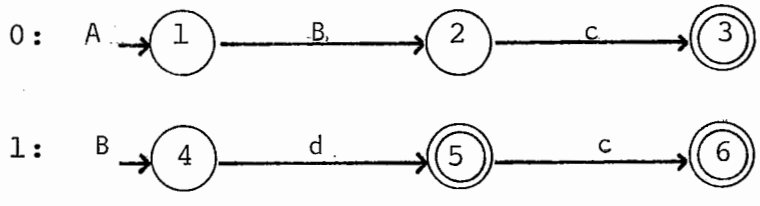


Esta gramática é LL(1) pois:

1 - Existe transição com c e B no mesmo estado

2 - c não está contido em FIRST (B)

(3) $A \rightarrow Bc$
$B \rightarrow d(c)^?$

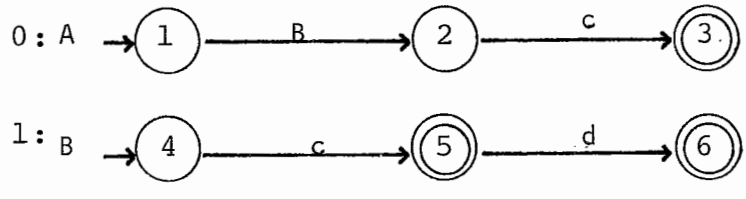


Esta gramática não é LL(1) pois:

1 - Existe transição com c em estado final de B

2 - c está contido em FOLLOW (B)

(4) $A \rightarrow Bc$
$B \rightarrow c\ (d)^?$



Esta gramática é LL(1) pois:

1 - Existe transição com d em estado final de B

2 - d não está contido em FOLLOW (B)

3. ÁRVORES BINÁRIAS

Uma árvore binária é definida como um conjunto T de zero ou mais nós tal que:

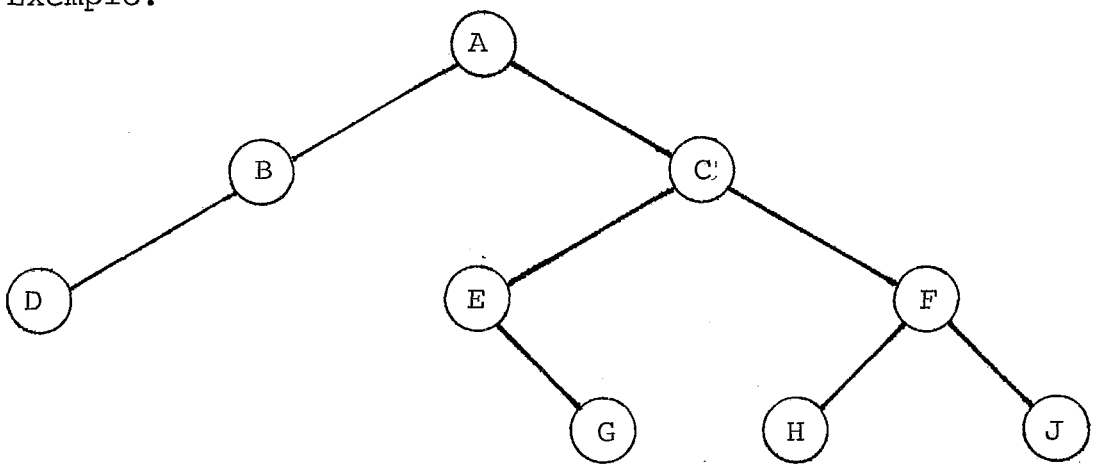
(1) Existe um nó especialmente designado de raiz da árvore T ou não existe nenhum nó; e

(2) Os nós restantes (excluindo a raiz) são particionados em $2 \geq m \geq 0$ conjuntos disjuntos T_1, T_m, e cada um destes conjuntos é por sua vez uma árvore binária. As árvores binárias T_1 T_m são chamadas sub-árvores da raíz.

Da definição, segue-se o fato de que todo nó de uma árvore binária é a raiz de alguma sub-árvore contida na árvore inteira. O número de sub-árvores de um nó é chamado de grau daquele nó. Um nó de grau zero é chamado de nó terminal ou de folha. Um nó de grau diferente de zero é chamado de nó interno.

Cada raiz é chamada de pai dos nós raízes de suas sub-árvores, e estes nós raízes são ditos irmãos entre si, e filhos de seu pai.

Exemplo:



Há 3 principais maneiras de se percorrer uma árvore binária, de modo que cada nó seja visitado exatamente uma vez:

- pré-ordem
- in-ordem
- pós-ordem.

Estes 3 métodos são definidos recursivamente:

(1) Quando a árvore binária é vazia, ela é percorrida sem que nada seja feito.

(2) Caso contrário, o percurso é feito através de 3 passos:

(a) percurso em pré-ordem:

I) Visite a raiz

II) Percorra a sub-árvore esquerda

III) Percorra a sub-árvore direita

(b) Percurso em in-ordem:

I) Percorra a sub-árvore esquerda

II) Visite a raiz

III) Percorra a sub-árvore direita

(c) Percurso em pós-ordem:

I) Percorra a sub-árvore esquerda

II) Percorra a sub-árvore direita

III) Visite a raiz

Um percurso completo através de uma árvore nos fornece um arranjo linear dos nós, de modo que é possível falar-se em nó sucessor ou nó predecessor de outro, em uma dada seqüência. Como será visto, isto será bastante útil.

Para a árvore do exemplo temos:

- Nós em pré-ordem : ABDCEGFHJ
- Nós em in-ordem : DBAEGCHFJ
- Nós em pós-ordem : DBGEHJFCA

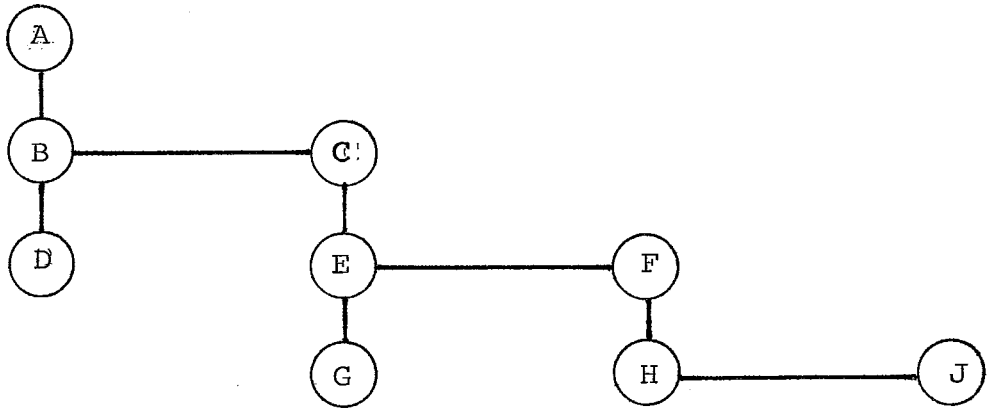
Diz-se, por exemplo, que o nó E é o sucessor do nó C em pré-ordem e o predecessor do nó H em pós-ordem.

Está provado que existe um método capaz de criar uma árvore binária que esteja em correspondência biunívoca com uma árvore qualquer ($m >= 0$).

Uma maneira de representar graficamente uma árvore binária é ligando-se filhos de um mesmo pai através de linhas horizontais e ligando-se um pai a seu primeiro filho, o filho primogênito, através de linhas verticais. Esta representação poder-se-ia chamar-

de árvore binária com ligações primogênito/irmão.

A mesma árvore já apresentada, nesta representação ficaria assim:



Dispondo-se apenas da lista de nós em uma dada ordem, não é possível reconstruir-se a árvore ou obter-se a lista dos nós em outra ordem.

Para que isto seja possível, é necessário ter-se informações sobre as ligações de cada nó. Isto é, informações sobre a raiz de cada sub-árvore.

Uma forma de agregar-se estas informações é representar-se cada nó de uma lista através de uma tripla (A, L_1, L_2), onde:

(1) A é o conteúdo do nó;

(2) L_1 representa a ligação do nó A com seu filho primogênito;

- Se $L_1 = \#$, então A tem filho, é um nó não-terminal
- Se $L_1 = \lambda$, então A não tem filho, é um nó terminal

(3) L_2 representa a ligação do nó A com o seu irmão imediatamente mais "novo":

- Se $L_2 = \#$, então A tem irmão
- Se $L_2 = \lambda$, então A não tem irmão.

Representando a árvore do exemplo por este método e colocando-se os nós em pré-ordem:

<A#λ>, <B##>, <Dλλ>, <C#λ>, <E##>, <Gλλ>, <F#λ>, <Hλ#>, <Jλλ>

Evidentemente, esta representação é redundante, já que os símbolos "#" podem ser substituídos pelos próprios nós que ocupam as ligações por eles indicados. No exemplo,

<A<B<Dλλ><C<E<Gλλ><F<Hλ<Jλλ>λ>>λ>>λ>

Além disso, ainda podemos diminuir mais a redundância representada pelo par "<>", usando apenas o indicador de raiz "<", já que o símbolo ">" não traz nenhuma informação adicional. No exemplo,

<A<B<Dλλ<C<E<Gλλ<F<Hλ<Jλλλλλ

Estas duas últimas representações, no entanto, vão necessitar de algorítmos mais sofisticados para o seu manuseio. Enquanto que a representação com redundâncias vai dar origem a algorítmos bem mais simples. Preferimos, então, utilizá-la.

Um algorítmo que será de bastante utilidade posteriormente obtem a partir da lista dos nós de uma árvore lida em pre-ordem, representada da maneira recém descrita, a lista dos nós da mesma árvore em pós-ordem.

Este algorítmo é bastante simples, necessitando apenas de uma pilha, inicialmente contendo o símbolo "$". A lista de nós em pré-ordem é lida da esquerda para a direita, um nó de cada vez. Para cada par (topo da pilha, símbolo na entrada), o algorítmo efetua um conjunto de ações.

O algorítmo termina quando não há mais nós a serem lidos e a pilha está vazia.

As ações efetuadas pelo algorítmo são as seguintes:

TOPO	ENTRADA	AÇÃO
λ	Qualquer	- Desempilha - Emite topo - Desempilha
$ ou A	<A##>	- Empilha A
$ ou A	<A#λ>	- Empilha λ - Empilha A
$ ou A	<Aλ#>	- Emite A
$ ou A	<Aλλ>	- Emite A - Emite topo - Desempilha

Usando a árvore do exemplo e sua representação em pré-ordem:

PILHA	ENTRADA	SAÍDA
$	<A#λ>	
$λA	<B##>	
$λAB	<Dλλ>	DB
$λA	<C#λ>	
$λAλC	<E##>	
$λAλCE	<Gλλ>	DBGE
$λAλC	<F#λ>	
$λAλCλF	<Hλ#>	DBGEH
$λAλCλF	<Jλλ>	DBGEHJF
$λAλCλ		DBGEHJFC
$λAλ		DBGEHJFCA
$λ		DBGEHJFCA$

4. ESQUEMA DE TRADUÇÃO DIRIGIDO PELA SINTAXE - SDTS

Seja Σ um alfabeto de entrada e Δ um alfabeto de saída. Define-se uma tradução de uma linguagem L_1 contida em Σ para uma linguagem L_2 contida em Δ como sendo a relação τ de Σ^* para Δ^* tal que o domínio de τ é L_1 e o contradomínio de τ é L_2.

Uma sentença y tal que (x,y) está em τ e dita uma saída para x.

Um dispositivo que, dada uma seqüência de entrada x, gere uma seqüência de símbolos de saída y, tal que (x,y) esteja numa dada tradução τ, é dito um tradutor para τ.

Um esquema de tradução dirigido pela sintaxe - o SDTS - é um formalismo para definir traduções. Intuitivamente, um SDTS é simplesmente uma gramática com elementos de tradução acoplados a cada produção. Sempre que uma produção é usada na derivação de uma sentença de entrada, o elemento de tradução é usado para ajudar a computar a porção da sentença de saída associada à porção da sentença de entrada gerada por aquela produção.

Um esquema de tradução T define uma tradução $\tau(T)$. Um tradutor para $\tau(T)$ pode ser construído de maneira tal que, dada uma seqüência de entrada x, o tradutor acha, se possível, alguma derivação de x a partir de S usando as produções do esquema de tradução.

Suponha que esta derivação seja:

$$S \Rightarrow \alpha_0 \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_n$$

Neste caso o tradutor cria uma derivação de formas de tradução:

$$(\alpha_0, \beta_0) \Rightarrow (\alpha_1, \beta_1) \Rightarrow \ldots \Rightarrow (\alpha_n, \beta_n),$$

sendo os α_i's formas sentenciais de entrada e os β_i's formas sentenciais de saída tal que:

- $(\alpha_0, \beta_0) = (S,S)$
- $(\alpha_n, \beta_n) = (x,y)$
- Cada β é obtido aplicando-se a β_{i-1} o elemento de tradução correspondente à produção usada na passagem desde α_{i-1} para α_i no ponto correspondente.

A seqüência y é uma saída para x.

Freqüentemente, é este o nosso caso, as formas sentenciais de saída podem ser criadas ao mesmo tempo em que se faz o "parse" da entrada.

Uma importante classe de esquemas de tradução é o SDTS - esquema de tradução dirigido pela sintaxe.

Um SDTS é uma quíntupla $T = (N,\Sigma,\Delta,R,S)$, onde

(1) N é um conjunto finito de símbolos não terminais

(2) Σ é um alfabeto finito de entrada

(3) Δ é um alfabeto finito de saída

(4) R é um conjunto finito de regras da forma $A \to \alpha,\beta$ onde,
- α pertence $(N \cup \Sigma)^*$
- β pertence $(N \cup \Delta)^*$
- Os não terminais em β são uma permutação dos não terminais em α.

(5) S é um não terminal particular em N, o símbolo de entrada.

A tradução definida por T, denotada $\tau(T)$, é o conjunto de pares:

$$\{(x,y) \mid (S,S) \overset{*}{\Rightarrow} (x,y),\ x \text{ pertence a } \Sigma^* \text{ e } y \text{ pertence a } \Delta^*\}$$

Se $T = (N,\Sigma,\Delta,R,S)$ é um SDTS, então $\tau(T)$ é dita uma tradução dirigida pela sintaxe (SDT).

A gramática $G_i = (N,\Sigma,P,S)$, onde $P = \{A \to \alpha \mid A \to \alpha,\beta \text{ está em } R\}$, é chamada gramática de entrada do SDTS T.

A gramática $G_0 = (N,\Delta,P',S)$, onde $P' = \{A \to \beta \mid A \to \alpha,\beta \text{ está em } R\}$, é chamada gramática de saída de T.

Um SDTS $T = (N, \Sigma, \Delta, R, S)$ tal que em cada regra $A \rightarrow \alpha, \beta$ em R, não terminais associados ocorram na mesma ordem em α e β é chamado SDTS simples e sua tradução de SDT simples. Tradutores para SDTS simples são fáceis de serem construídos e, em nosso caso, cuidados foram tomados na definição da gramática de saída para que um SDTS simples pudesse ser usado.

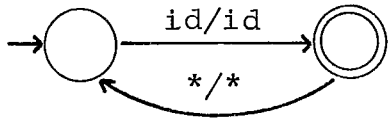
O tradutor mais simples é o transdutor finito, composto por um reconhecedor acoplado a um transmissor. O transdutor finito já foi definido e descrito no capítulo IV, durante a discussão da análise léxica. Repetiremos, por conveniência, a definição.

Um transdutor finito é uma 6-Tupla $(Q, \Sigma, \Delta, \delta, q_0, F)$ onde,

(1) Q é um conjunto finito de estados
(2) Σ é um alfabeto de entrada
(3) Δ é um alfabeto de saída
(4) δ é uma função de transição δ: $(Q x \Sigma) \rightarrow \mathcal{P}(Q \times \Delta^*)$
(5) q_0 pertence a Q, sendo o estado inicial
(6) F está contido em Q, sendo o conjunto de estados finais.

δ pode ser definido por um grafo de transição, em que cada aresta possui um rótulo do tipo x/y. Um rótulo x/y numa aresta dirigida do nó q_i para o nó q_j significa que $\delta(q_i, x)$ contém (q_j, y), isto é, existe uma transição válida do estado q_i para o estado q_j com o símbolo de entrada x, e durante a transição é emitida a seqüência de símbolos y.

EXEMPLO:



5. DENDRO-TRADUTOR RRP

Para obter-se um transdutor finito que emita árvores binárias na saída, define-se um dendro-transdutor finito como sendo uma 6-Tupla $DT = (Q,\Sigma,\Delta,\delta,q_0,F)$, onde todos os elementos têm o mesmo valor que na definição de transdutor finito, com exceção de:

(4) δ é um mapeamento de $(Q x \Sigma) \rightarrow (Q x \Theta)$, onde Θ é um elemento do universo das árvores binárias rotuladas com símbolos de Δ.

Definindo-se um esquema de tradução dirigido pela sintaxe para gramáticas de entrada RRP:

Um SDTS-RRP é definido como uma 5-TUPLA (N,Σ,Δ,P,S) onde:

(1) N é um conjunto finito de símbolos não terminais

(2) Σ é um alfabeto finito de entrada

(3) Δ é um alfabeto finito de saída

(4) P é um conjunto de produções da forma $A \rightarrow T$, onde T é um transdutor finito $T = (Q,(N \cup \Sigma),(N \cup \Delta),\delta,q_0,F)$

(5) S é um não terminal particular em N, o símbolo de entrada.

Combinando-se as duas últimas definições para obter-se um esquema de tradução dirigido pela sintaxe para gramáticas de entrada-RRP que emita árvores binárias na saída:

Um dendro-tradutor RRP é definido como sendo uma 5-TUPLA (N,Σ,Δ,P,S) onde:

(1) N é um conjunto finito de símbolos não terminais

(2) Σ é um alfabeto finito de entrada

(3) Δ é um alfabeto finito de saída

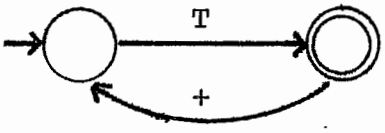
(4) P é um conjunto de produções da forma $A \rightarrow DT$, onde DT é um dendro-transdutor finito

$$DT = (Q,(N \cup \Sigma),(N \cup \Delta),\delta,q_0,F)$$

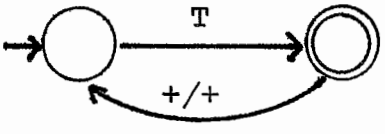
(5) S é um não terminal particular em N, o símbolo de entrada.

Exemplificando os diversos conceitos introduzidos:

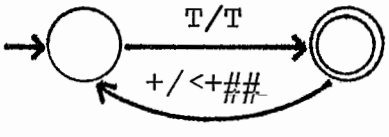
1) Autômato finito determinístico



2) Transdutor finito



3) Dendro-transdutor finito



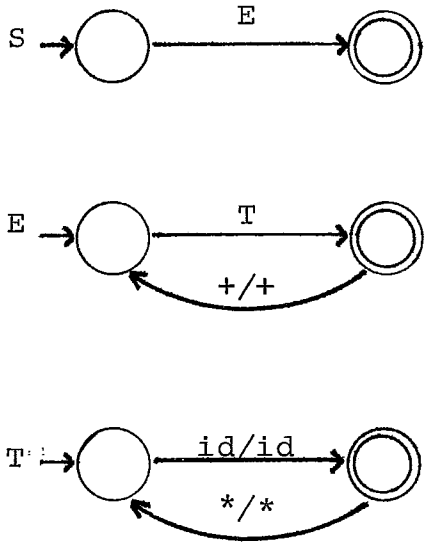
4) SDTS

E → E + T , E T +

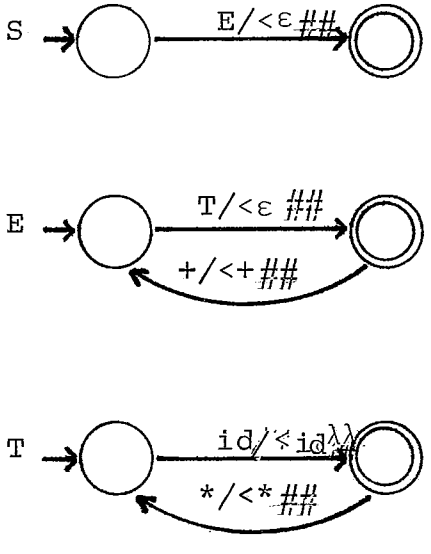
T → T * id, T id *

id , id

5) SDTS RRP



6) Dendro-tradutor RRP



VI. ANÁLISE SINTÁTICA/GERAÇÃO DE CÓDIGO. IMPLEMENTAÇÃO

1. ANALISADOR SINTÁTICO

A linguagem Loban foi projetada por uma equipe de especialistas em banco de dados. Conforme é destacado em {1}, "a filosofia usada na definição da interface Loban é basicamente a separação da interface do sistema portador (hardware + software básico), portanto toda a interface foi definida a nível funcional sem considerar nenhum aspecto da implementação".

A mesma idéia, considerada fundamental na especificação do Loban já apareceu nos capítulos 2 e 3 .

Este enfoque, bastante valioso sob o ponto de vista de implementação de estruturas de banco de dados, causa óbvias dificuldades para a análise sintática, pois o conceito de definição da interface "sem considerar nenhum aspecto da implementação" foi estendido até a sintaxe.

Por exemplo, na primeira especificação recebida o caracter "/" (barra) era usado de forma ambígua, aparecendo em contextos semanticamente diferentes:

1 - Associado ao conceito de marcação de endereço de ponto
2 - Separador dos componentes de um literal representando uma data (dia/mês/ano)
3 - Operador aritmético de divisão

Este caso e outros semelhantes dificultavam sobremaneira o uso de um método simples para a análise sintática.

Paralelamente a isso, a definição do Microloban a partir do Loban foi um processo que estava em andamento durante todo o tempo em que o analisador estava sendo desenvolvido. Até se "fechar a sintaxe", houve diversas versões da gramática Microloban. Algumas modificações foram forçadas pelo projetista do analisador, devido a problemas como o descrito acima e outras foram definidas pe

los projetistas da linguagem.

Houve necessidade de um método em que a gramática usada pelo analisador fosse o máximo possível semelhante a gramática definida pelos usuários, em sua maioria leigos em teoria de compiladores, para que uma pequena mudança que fosse necessária na sintaxe fosse facilmente efetuada na gramática do interpretador.

A primeira tentativa foi escrever-se uma gramática LL(1) para a linguagem Microloban e projetar-se um analisador descendente determinístico em um passo e com um símbolo de avanço. Conquanto o analisador resultasse simples, descobrir uma gramática LL(1) era bastante complicado, se é que esta existisse.

Na primeira versão havia 383 produções e 112 não terminais, 84 deles sem qualquer significado semântico. A pretensa gramática LL(1) não guardava nenhuma semelhança com a do usuário. Além disso, quando submetida a um programa gerador de analisadores sintáticos LL(1), foram constatados 38 pontos na gramática onde a condição LL(1) era violada (conflitos), isto é, a gramática ainda não era LL(1). A tentativa de solução dos conflitos revelou-se uma tarefa praticamente impossível. O principal problema estava na falsa recursão, que "introduz na sintaxe uma inadequação do mecanismo descritivo que obscurece sua compreensão" {3}.

Em {4}, estão descritas, de um ponto de vista genérico, as dificuldades encontradas no uso da técnica LL(1) e conclue-se pela necessidade de "um formalismo que reflete mais exatamente a maneira pela qual concebemos as coisas e que é mais conveniente que um artificial".

Em busca de uma solução que refletisse a gramática natural chegou-se ao método RRP LL(1), isto é, uso de um analisador sintático descendente, a partir de uma gramática de entrada com lados direitos regulares.

Este caminho foi sugerido por {3} e {5}. Citando-se mais uma vez: "gramáticas que possuem conjuntos regulares como lados direitos de suas produções vem sendo intensamente estudadas nos últimos anos por representarem uma maneira mais natural de se especificar linguagens livres de contexto. É comum encontrarmos definições de linguagens escritas na forma de diagramas sintáticos" ({5} e suas referências).

O uso de expressões regulares para representar a sintaxe Microloban foi extremamente bem sucedido, passando a ser utilizado pela equipe de banco de dados em seus contatos com o autor deste trabalho.

A gramática possui apenas 16 não terminais, cada um com um significado semântico bem definido em Loban. Além disso, as modificações na sintaxe tornaram-se bem menos dolorosas.

A construção de uma tabela de controle para o analisador sintático, a partir da representação da gramática RRP em formato de gramática com os lados direitos representados por automato finito determinístico, é muito simples, como será visto.

A tarefa de transformar-se a gramática escrita em forma de expressão regular para a forma de lados direitos representados por automato finito determinístico foi efetuada automaticamente, utilizando-se programas já disponíveis no B-6700 do núcleo de computação eletrônica da UFRJ, desenvolvidos como parte do projeto NHAONHAO da COPPE/UFRJ ({7} e {12}).

Inicialmente, usou-se o codificador RRP, que lê a gramática escrita em forma de expressão regular e gera um arquivo em disco, com a gramática em forma de árvore binária costurada. O codificador gera também uma listagem do arquivo de entrada, mostrada no Apêndice 3, junto com uma lista dos terminais e não terminais, e os códigos numéricos a eles atribuídos pelo codificador.

Na prática, o usuário não toma conhecimento do formato arvo-

re binária, pois este arquivo é usado apenas como entrada do programa denominado "alterador RRP". Este gera uma listagem da gramática em formato com lados direitos representados por automatos - finitos determinísticos, mostrada no Apêndice 4.

No final desta seção estão exemplos: primeiro, da gramática em forma de expressão regular, conforme produzida pelo programa codificador RRP; segundo, da gramática com lados direitos em forma de AFD, conforme produzida pelo programa alterador.

A verificação das duas condições para que uma gramática seja LL (1) numa gramática do porte da do Microloban é uma tarefa de fôlego. Até o momento do término do desenvolvimento deste trabalho ainda não havia na UFRJ um programa que efetuasse estas tarefas automaticamente. O trabalho foi, portanto, realizado manualmente.

O Apêndice 2 mostra a computação dos FIRST's e FOLLOW's dos não terminais da gramática. No Apêndice 4, que mostra a saída do programa alterador RRP, estão assinalados os pontos que ainda violam a segunda condição.

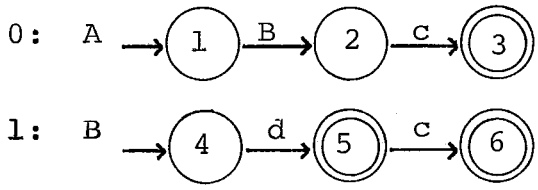
Durante o processo de "fechar a sintaxe" havia diversos pontos em que uma ou outra das condições era violada. A gramática foi reescrita e, em alguns casos, a sintaxe foi modificada pelo usuário. Todos os problemas relativos à primeira condição foram desta forma solucionados.

Mas alguns casos em que a segunda condição não era obedecida não puderam ser resolvidos. A solução encontrada foi estabelecer a seguinte regra de precedência para o analisador:

- Nos casos em que houver conflito num possível estado final, o analisador optará por continuar na mesma produção.

EXEMPLO:

A→B c
B→d(c)?



Assim, no caso do exemplo acima, ao encontrar dc na entrada, o analisador fará as transições 1→4→5→6, ao invés de fazer as transições 1→4→5→2.

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NUCLEO DE COMPUTAÇÃO ELETRÔNICA

```
==========================================================================================================
                MODULO CODIFICADOR DE EXPRESSOES REGULARES                                  FOLHA : 1
==========================================================================================================
LN PR                          IMAGEM DO CARTAO                          NAO TERMINAIS        TERMINAIS
                                                                          NOME    COD.         NOME    COD.
1  1   TRABALHO = ('EXECUTAR'/'COMPILAR') 'PARA' 'USUARIO' 'ID'
                                                                          TRABALHO  200        EXECUTAR  1
                                                                                               COMPILAR  2
                                                                                               PARA      3
                                                                                               USUARIO   4
                                                                                               ID        5
2  1              'PAR-ABRE' 'ID' 'PAR-FECHA'
                                                                                               PAR-ABRE  6
                                                                                               PAR-FECHA 7
3  1                ('ID'/'INSTRUCOES') ':' ( COMANDO   ';' )* 'ENCERRAR' ;
                                                                                               INSTRUCOES 8
                                                                          COMANDO   201        :         9
                                                                                               ;         10
                                                                                               ENCERRAR  11
4  2   COMANDO = ('CRIAR' 'ACSET' 'ID' ( 'A' 'PARTIR' 'DE' 'ACSET' 'ID'
                                                                                               CRIAR     12
                                                                                               ACSET     13
                                                                                               A         14
                                                                                               PARTIR    15
                                                                                               DE        16
5  2              'ATE' 'VERSAO' 'NINT' )? /
                                                                                               ATE       17
                                                                                               VERSAO    18
                                                                                               NINT      19
6  2              'ABRIR' 'ACSET' 'ID' ('COM' 'ID' & ',')?)';'
                                                                                               ABRIR     20
                                                                                               COM       21
                                                                                               ,         22
7  2              ('ABRIR' 'VOLUME' 'DE' ('ENTRADA'/'SAIDA'/'INTERACAO')
                                                                                               VOLUME    23
                                                                                               ENTRADA   24
                                                                                               SAIDA     25
                                                                                               INTERACAO 26
8  2              'ID' ('TECLADO'/'VIDEO'/'IMPRESSORA'/'FITA'/'DISCO'/
                                                                                               TECLADO   27
                                                                                               VIDEO     28
                                                                                               IMPRESSORA 29
                                                                                               FITA      30
                                                                                               DISCO     31
9  2                  'TELEIMPRESSORA') ';' )*
                                                                                               TELEIMPRESSO 32
10 2              'ESTABELECER' 'PROTECAO' 'ID' ('PARA' ('LER' /'ALTERAR')
                                                                                               ESTABELECER 33
                                                                                               PROTECAO  34
                                                                                               LER       35
                                                                                               ALTERAR   36
11 2                  'SOBRE' ('ID'/'ACTRAB') & 'U' ) & 'E'
                                                                                               SOBRE     37
```

```
AUTOMATO PRODUCAO #    1
  ESTADO :    1
    SIMBOLO    1 - TRANSICAO    2
    SIMBOLO    2 - TRANSICAO    2
  ESTADO :    2
    SIMBOLO    3 - TRANSICAO    3
  ESTADO :    3
    SIMBOLO    4 - TRANSICAO    4
  ESTADO :    4
    SIMBOLO    5 - TRANSICAO    5
  ESTADO :    5
    SIMBOLO    6 - TRANSICAO    6
  ESTADO :    6
    SIMBOLO    5 - TRANSICAO    7
  ESTADO :    7
    SIMBOLO    7 - TRANSICAO    8
  ESTADO :    8
    SIMBOLO    5 - TRANSICAO    9
    SIMBOLO    8 - TRANSICAO    9
  ESTADO :    9
    SIMBOLO    9 - TRANSICAO   10
  ESTADO :   10
    SIMBOLO   11 - TRANSICAO   11
    SIMBOLO  201 - TRANSICAO   12
  ESTADO :   11F
  ESTADO :   12
    SIMBOLO   10 - TRANSICAO   10
```

```
AUTOMATO MINIMO
  ESTADO :    1
    SIMBOLO    1 - TRANSICAO    2
    SIMBOLO    2 - TRANSICAO    2
  ESTADO :    2
    SIMBOLO    3 - TRANSICAO    3
  ESTADO :    3
    SIMBOLO    4 - TRANSICAO    4
  ESTADO :    4
    SIMBOLO    5 - TRANSICAO    5
  ESTADO :    5
    SIMBOLO    6 - TRANSICAO    6
  ESTADO :    6
    SIMBOLO    5 - TRANSICAO    7
  ESTADO :    7
    SIMBOLO    7 - TRANSICAO    8
  ESTADO :    8
    SIMBOLO    5 - TRANSICAO    9
    SIMBOLO    8 - TRANSICAO    9
  ESTADO :    9
    SIMBOLO    9 - TRANSICAO   10
  ESTADO :   10
    SIMBOLO   11 - TRANSICAO   11
    SIMBOLO  201 - TRANSICAO   12
  ESTADO :   11F
  ESTADO :   12
    SIMBOLO   10 - TRANSICAO   10
```

EXEMPLO DA SAÍDA DO ALTERADOR

2. CÓDIGO INTERMEDIÁRIO

Como já foi visto, a saída do Dendro-Tradutor RRP é uma árvore binária. Historicamente, código intermediário em forma de árvore não tem sido diretamente implementado porque sempre se deu ênfase à necessidade dos compiladores serem rápidos, pequenos e de um passo. No entanto, o modelo de árvore esteve presente, mesmo que indiretamente.

Já atualmente, código intermediário em forma de árvore começa a ser usado, a partir do instante em que não só a velocidade e o tamanho do compilador são importantes, mas há também outras qualidades a considerar.

O uso de métodos formais de análise sintática já permite a alteração na sintaxe da linguagem sendo compilada sem impacto na lógica do compilador, pois apenas a tabela do analisador é afetada. O mesmo pode ser feito com relação à geração de código, através do uso de métodos formais para a especificação do código intermediário. Compiladores construídos seguindo este enfoque são muito mais fáceis de depurar e alterar, representam um progresso em relação aos compiladores artesanais e justificam plenamente os esforços de pesquisa empreendidos nesta área.

Em linguagens de programação os atributos de um identificador são definidos explicitamente no programa, ou então não são definidos em lugar algum, variando dinamicamente, mas sempre em função das operações com ele executadas ao longo do programa.

Em linguagens de banco de dados, isto já não ocorre. As definições dos tipos dos dados podem ser anteriores ao programa que as usa, estando armazenadas previamente na base de dados.

Para o tradutor realizar a análise semântica estática de um programa, dois enfoques seriam possíveis:

(1) O tradutor acessa a base de dados para realizar a análise semântica.
(2) A análise semântica estática é postergada para a fase de interpretação.

Se adotado, o primeiro enforque inviabilizaria a operação do tradutor para o processamento por lotes, pois as definições dos dados na base são dinâmicas e nada garante que um determinado-identificador teria a mesma definição no momento em que o código intermediário estivesse sendo gerado pelo tradutor, e posteriormente, quando este código fosse executado.

É conveniente recordar que Microloban é uma linguagem auto-contida que engloba comandos de definição, manipulação e gerência de dados, não utilizando nenhuma outra linguagem para a geração da base de dados. Um programa qualquer pode alterar os tipos de dados durante a sua execução.

A solução usada foi realizar a análise semântica estática durante a interpretação, de uma maneira simples: o código que vai efetuar a análise semântica está contido nas árvores a serem passadas para o interpretador, que o insere, por sua vez, no código executável.

Deste modo, conseguiu-se reduzir a interface entre o tradutor-e o interpretador unicamente às árvores de código, e mais nada.

Uma conclusão importante, considerando-se o fato dos tipos de dados serem dinâmicos em banco de dados, é que gramáticas de atributos não podem ser diretamente usadas em compiladores desse tipo de linguagem, se houver processamento em lotes.

As árvores de código associadas à sintaxe do Microloban estão representadas com os nós em pré-ordem.

A proposta incial era usar-se árvore como interface entre o

tradutor e o interpretador, que poderia dispor da árvore conforme suas necessidades.

Verificou-se posteriormente que nenhuma simplificação significativa era obtida deste modo, pois o interpretador não utiliza qualquer algoritmo que necessite do código intermediário em forma de árvore (por exemplo, não efetua otimizações), embora isso seja comum em compilação de linguagens de programação.

Neste trabalho, portanto, desenvolveu-se um tradutor dirigido pela sintaxe para gramáticas com lados direitos regulares gerando árvores como saída. Este esquema é um processo teórico significativo para uso em compilação. Na aplicação para Microloban,o esquema desenvolvido foi mais poderoso que o necessário.

Deste modo, optou-se por introduzir um módulo, que podemos chamar de linearizador, após a geração da árvore pelo tradutor,que com a ajuda de uma pilha, gera a lista dos nós da árvore percorrida em pós-ordem. O linearizador utiliza o algoritmo descrito na Seção 3 do Capítulo V, com uma pequena modificação, descrita mais adiante.

A interface entre o tradutor e o interpretador passa, portanto, a ser uma lista linear de nós que será tratada pelo interpretador como uma pilha. Entretanto, toda a especificação do tradutor e do interpretador foi feita sobre um modelo em árvore do código intermediário.

O uso deste modelo foi extremamente feliz, por permitir uma visualização da seqüência de execução dos comandos de um programa Microloban, visualização esta que uma simples lista linear de comandos, não permitiria. Por causa disso, o modelo serviu como um meio de comunicação eficaz entre o autor desta tese e a equipe do projeto Miniban, facilitando sobremaneira o trabalho.

Nenhuma informação adicional sobre as ligações entre os nós é passada para o interpretador, além da seqüência de nós. Todas as informações semânticas e de controle de execução de comandos estão agregadas, pela própria maneira como o código foi definido, ao conteúdo de cada nó, ou seja, a tradução está totalmente contida na própria árvore.

A definição do código intermediário e a construção do interpretador são os temas de {10}. A especificação do código intermediário em forma de árvore, isto é, a definição da porção de código intermediário a ser emitida pelo dendro-tradutor RRP a cada transição está exemplificada no Apêndice 5.

Para uma boa visualização, os nós das árvores foram representados graficamente do seguinte modo:

(1)	<A##>	Nó não terminal com irmão	A
(2)	<A#λ>	Nó não terminal sem irmão	A
(3)	<Aλ#>	Nó terminal com irmão	A
(4)	<Aλλ>	Nó terminal sem irmão	A
(5)	λ	Nó λ	λ

O último tipo de nó foi incluído na definição para indicar a ausência de irmão nos casos em que não se pode estabelecer a priori se um nó tinha ou não irmão. Quando um nó deste tipo estiver incluído na porção de código intermediário gerada, o interpretador saberá que se trata do final de uma produção.

e que a última ligação deixada pendente, para um possível irmão, na verdade não existe.

A representação usada em {10} é repetida no Apêndice 5 não é funcional como auxílio à implementação, pois mostra o código - intermediário associado à gramática de entrada escrita em formato de lados direitos regulares.

Foi necessário reescrever-se a gramática em formato de lados - direitos como automatos finitos determinísticos, com os nós de código intermediário associados a cada transição dos auto - matos. Esta representação é exemplificada no apêndice 6.

Como já dito, o mesmo algoritmo descrito na Seção V.3 foi usado para a implementação do linearizador. Apenas uma pequena modificação foi feita para prever a existência dos nós λ.

Ao ser encontrado um nó tipo λ na entrada, as seguintes ações são executadas:

TOPO	ENTRADA	AÇÃO
A	λ	- Emite Topo - Desempilha - Empilha

Pelas próprias características do algoritmo, sempre que há um nó λ na entrada, haverá um nó A no topo da pilha.

Os nós gerados pelo linearizador para o interpretador não são todos do mesmo tipo. Alguns representam instruções a serem executadas. Outros trazem informações semânticas, tais como o valor associado a um literal numérico, por exemplo. Nós incluídos neste segundo caso são sempre nós terminais, ou folha, da árvore que foi linearizada.

Deste modo, alguns nós terão comprimentos maiores que os ou-

tros, por conterem mais informações a serem passadas para o interpretador.

A interface entre o tradutor e o interpretador é uma lista linear de nós. O tradutor coloca os nós gerados em seu final e o interpretador tira os do início para execução.

Na especificação do código intermediário foi incluído ainda um tipo de nó especial, o nó "EXECUTE". Este tipo de nó é uma instrução de controle para o tradutor, indicando o momento de passar o controle ao interpretador.

3. FUNCIONAMENTO DO SDTS

Devido ao uso de um esquema de tradução dirigido pela sintaxe (SDTS), a geração de código ocorre simultaneamente com a análise sintática, e o funcionamento desta última é influenciado, em muitos aspectos, pela primeira, tornando-se impossível dissociá-los.

Numa implementação completa do interpretador, o analisador sintático é chamado pelo interpretador sempre que este necessitar de mais um conjunto de ações a executar.

O analisador sintático solicitará itens sintáticos ao analisador léxico e, usando-os como entrada, efetuará transições estado a estado, gerando durante este processo nós correspondentes a árvore de código em pós-ordem a ser passada para o interpretador.

O controle será devolvido pelo analisador ao interpretador quando, no processo de geração de código, for gerado um nó do tipo "EXECUTE". O interpretador percorrerá a seqüência de nós recebida, efetuando as ações indicadas, e novamente retornará o comando ao analisador sintático, solicitando mais.

A existência do nó "EXECUTE" não é devida a qualquer exigência do esquema de tradução proposto nesta tese. Na verdade, o tradutor poderia devolver o comando ao interpretador em qualquer ponto. Este nó é necessário devido às exigências de recuperação do estado do interpretador em caso de erro sintático, nos modos interativo ou lote.

Para o teste do programa tradutor elaborado nesta tese, o interpretador é simulado por uma rotina que apenas gera uma saída impressa, contendo a seqüência de código gerada.

Para fins de documentação também é gravado um arquivo em disco com a árvore representada com os nós de código em pré-ordem, para permitir a reconstrução da árvore.

Este arquivo é lido pelo programa desenhador de árvores, descrito no Capítulo VII, que gera uma representação gráfica das árvores, com ligações primogênito/irmão, para melhor visualização.

A tabela de controle do analisador, conforme já explicado, é a própria tabela (ou AFD) obtida pelo gerador de analisadores sintáticos RRP LL(1), mostrada no Apêndice 3.

A gramática RRP LL(1) do Microloban deu origem a uma tabela de 306 estados versus 201 terminais e 16 não terminais, num total de 306 x (201 + 16) = 66402 combinações. Apenas cerca de 1% destas combinações são válidas, o que caracteriza uma matriz bastante esparsa.

Deste modo, a implementação da maneira mais simples, em forma tabular é inviável, sendo necessário adotar-se um formato de representação mais sofisticado, que utilize uma quantidade de memória diretamente proporcional ao número de combinações válidas e não- ao número total de combinações.

Foi adotada a representação em forma de lista, cujos elementos - representam os pares (estado atual, símbolo na entrada) para os quais há transições válidas. Deste modo, cada estado da tabela é representado por um conjunto de elementos, um para cada símbolo- com o qual haja uma transição válida a partir daquele estado. No algorítmo, "ir para um estado" significa transferir o ponteiro - da lista para o primeiro elemento associado àquele estado. Cada- elemento da lista é uma quadrupla da forma: (simbentra, sucessor, alternativa, ação) onde,

SIMBENTRA é o código de um símbolo terminal (quando ação é 1) ou um apontador para uma lista de first de um não terminal (quando ação é 2).

SUCESSOR é a posição na lista de estados da primeira quadrupla associada ao estado sucessor.

ALTERNATIVA — é um campo que indica se aquela quádrupla é a última referente aquele estado (alternativa=0) ou se a quádrupla seguinte também pertence ao mesmo estado (Alternativa=1).

AÇÃO — é um código indicando a ação a ser tomada durante a transição associada a esta quádrupla:

Ação = 1 - Vá para o estado sucessor

Ação = 2 - O símbolo na entrada pertence ao first de um não terminal. Empilhe o estado atual na pilha de trabalho e entre na produção referente a este não terminal.

Ação = 3 - Estado final. Coloque o estado que está no topo da pilha de trabalho em estado atual.

EXEMPLO: Produção numero 1 da gramatica do Microloban escrita em forma de lista

POSIÇÃO	SIMBENTRA	SUCESSOR	ALTERNATIVA	AÇÃO
0	1	2	1	1
1	2	2	0	1
2	3	3	0	1
3	4	4	0	1
4	5	5	0	1
5	6	6	0	1
6	5	7	0	1
7	7	8	0	1
8	5	10	0	1
9	8	10	0	1
10	9	11	0	1
11	11	13	1	1
12	14	14	0	2
13	0	0	0	3
14	10	11	0	1

A passagem do formato de AFD para o formato lista foi feita manualmente, gerando uma grande carga de trabalho para o projetista do analisador, por não haver disponível na UFRJ um programa para fazê-lo automaticamente.

4. RECUPERAÇÃO DE ERROS

Os esquemas adotados para a recuperação de erros sintáticos no processamento em lotes e no processamento interativo são forçosamente diferentes.

Enquanto que no processamento por lotes o programa fonte está todo disponível a frente do ponto em que o erro ocorreu até o final do programa, no processamento interativo, o tradutor só dispõe da linha onde o erro ocorreu.

Em linguagens de acesso a banco de dados não pode existir um esquema sofisticado de reparação automática de erros sintáticos, já que seria potencialmente nocivo se um programa com erro rodasse sobre a base, mesmo que este erro tivesse sido reparado durante a análise sintática. Nenhuma garantia haveria que um programa neste caso causasse sobre os dados o efeito desejado pelo seu autor.

Optou-se, então, por um esquema simples de recuperação de erro por eliminação de frase, adaptado a cada um dos modos de processamento, que apresenta ainda as vantagens de ser rápido, ocupar pouco espaço e ser de fácil programação.

No processamento interativo era importante garantir que o conteúdo da base de dados não fosse alterado enquanto houvesse possibilidade de que o comando sendo digitado tivesse um êrro sintático, ou seja, que o comando não fosse passado para o interpretador enquanto não fosse seguro fazê-lo.

A gramática da linguagem foi estudada e identificados os pontos em que uma ação sobre a base de dados pode ser considerada completa. Nas árvores de código intermediário, isto é, na gramática de saída do tradutor foram colocados nós especiais de controle, chamados nós "EXECUTE".

O tradutor, ao processar um programa que vai sendo digitado durante uma sessão de terminal, vai armazenando a lista de comandos gerada e, só ao encontrar um nó deste tipo, passa o comando ao interpretador.

Imediatamente antes de passar o comando, o tradutor faz um registro do conteúdo de todas as variáveis que definem o seu estado corrente, sobrepondo-se a algum registro que porventura já tenha sido feito.

O interpretador executa as ações devidas e retorna o comando ao tradutor, que começa a processar uma nova linha de programa sendo digitada.

No caso desta linha conter um erro sintático, o tradutor executa as seguintes ações:

- Elimina da lista de comandos a ser passada ao interpretador todos os comandos gerados após o último nó "EXECUTE".

- Restaura o seu estado para o estado definido pelo registro feito no momento em que foi processado o último nó "EXECUTE".

- Comunica o fato ao usuário através de mensagem na tela.

- Solicita a digitação de uma nova linha de programa, ignorando a linha com erro.

É importante ressaltar que este esquema só pode ser implementado porque a linguagem o permitiu, pela simplicidade de suas estruturas de controle. A estrutura mais complicada é um tipo de iteração, o comando "fazer para", que ainda assim é compatível com o enfoque adotado.

No processamento em lotes, foi usado um esquema que pode ser considerado análogo ao usado no processamento interativo, guardadas as diferenças entre os dois casos.

Não é possível, em caso de erro sintático no processamento em lotes, usar-se o nó "EXECUTE" porque é necessário processar o restante do programa, à frente do ponto de erro, sem que seja possível descobrir-se onde seria gerado o próximo nó deste tipo, já que o analisador está num estado sintático inválido.

É necessário, neste caso, utilizar-se algo explícito na sintaxe do programa - o delimitador de final de comando em Microloban, o caracter ";" (ponto-e-vírgula) {13}.

Toda vez que este caracter é encontrado no programa sendo lido, o tradutor efetua um registro de seu estado corrente, sobrepondo-se a algum registro anteriormente efetuado.

O comando continua sendo passado ao interpretador quando um nó "EXECUTE" é gerado, mas no processamento em lotes, o interpretador apenas armazena a lista de comandos para processamento posterior, sem efetuar qualquer ação sobre a base de dados.

Deste modo, a recuperação de erros sintáticos se destina unicamente a permitir a análise sintática de um programa errado até o final. A condição de erro é comunicada ao interpretador para que o programa em questão não venha jamais a ser executado. O interpretador retorna o comando ao tradutor que continua processando as linhas do programa.

Em caso de erro sintático, o tradutor executa as seguintes ações:

- Restaura o seu estado para o estado definido pelo registro feito no momento em que foi processado o último caracter ";" no texto do programa.

- Liga um indicador de erro para que o interpretador nas próximas vezes que for chamado ignore todo o código gerado.

- Lê os próximos registros (linhas) do programa fonte até que um caracter ";" seja encontrado, ignorando todos os caracteres entre o ponto de erro e o ";".

- Emite uma mensagem de erro no mesmo dispositivo de saída onde o programa está sendo listado.

- Recomeça a análise sintática no primeiro caracter após o ";" esperando que seja compatível com o novo estado corrente do tradutor.

Pela própria estrutura do Microloban, com um caracter ";" delimitando todo final de comando, este enfoque deve ter sucesso em boa parte dos casos.

Convém lembrar que o tradutor não efetua nenhuma espécie de análise semântica, garantindo apenas a correção sintática de um programa.

VII. UTILIZAÇÃO

Dentro do conceito do Microloban como uma linguagem autocontida e independente do sistema portador, o tradutor foi implementado de maneira que o usuário necessite apenas chamá-lo após abrir a sessão de terminal. Nenhum comando ou diretiva do COBRA-300 é executado, tal como atribuir uma unidade lógica a um arquivo, - que é normalmente necessário neste tipo de máquina.

Todas as diretivas que o usuário precise comunicar ao sistema são feitas em Microloban.

Inicialmente, o usuário chama o programa através do qual o tradutor foi implementado, denominado MLOBAN. A instrução inicial- de qualquer programa é sempre digitada via terminal, e nela são passadas as informações de identificação do usuário e modo de processamento, que em outros sistemas são comumente fornecidas- através de uma linguagem de controle.

A sintaxe desta instrução, usando a representação BNF, é a seguinte:

(EXECUTAR|COMPILAR) PARA USUÁRIO identificador$_1$
"("identificador $_2$")" (INSTRUÇÕES|identificador$_3$) ":"

A definição se se trata de processamento interativo ou em lotes é feita através da seguinte opção:

- Se for digitada a palavra reservada INSTRUÇÕES, o tradutor- assume que todo o programa vai ser digitado via terminal, em processamento interativo.

- Caso contrário, se for digitado um identificador, o tradutor sabe que se trata de processamento em lotes e vai ler o restante do programa num arquivo já existente, cujo nome é dado pelo identificador digitado.

Os outros dois identificadores são utilizados pelo interpreta - dor para identificar o usuário e verificar se ele está autoriza do a executar as funções que venha a solicitar no decorrer do programa.

O usuário dispõe ainda da opção de executar o programa ou ape - nas compilá-lo, para uma execução posterior.

Na Figura VII.1 está representado o fluxo de processamento do tradutor.

As saídas produzidas pelo tradutor são as seguintes:

- Arquivo em disco COER, contendo descrições dos tipos de construção que compõem a base de dados, chamadas verbetes de coerência. Este arquivo nem sempre é criado, dependendo do programa Microloban.

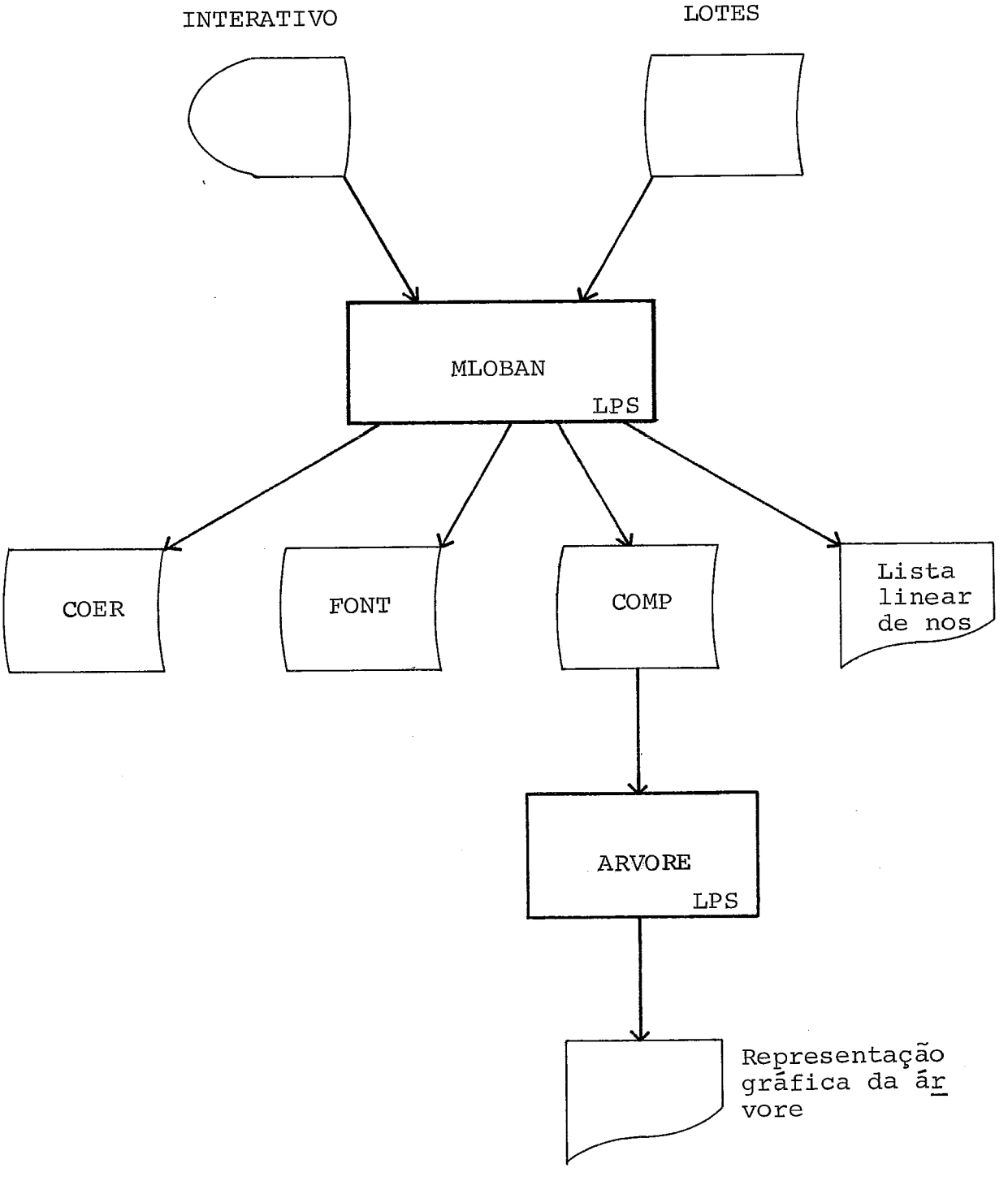
- Arquivo em disco FONT, contendo trechos do programa fonte sendo lido. Este arquivo também só será criado se o programa o pedir. Posteriormente, este arquivo pode ser lido pelo tradutor, durante uma sessão de processamento em lotes.

- Saída impressa, com a lista dos nós da árvore de comandos gerada, percorrida em pós-ordem. Este relatório simula a interface do tradutor com o interpretador. Quando o interpretador for implementado, esta saída impressa não mais será gerada.

- Arquivo em disco COMP, contendo a árvore de comandos com os nós de código em pré-ordem. Este arquivo será lido pelo pro - grama ÁRVORE, que gera uma representação gráfica da árvore.

Fig. VII.1

Fluxo de Processamento



O programa ÁRVORE foi escrito para permitir uma melhor visualização e entendimento do código gerado. Ele foi indispensável para a depuração do tradutor e ainda será extremamente útil para duas finalidades, enquanto o interpretador não for implementado:

- Avaliar a especificação do código intermediário

- Permitir a crítica de programas escritos em Microloban.

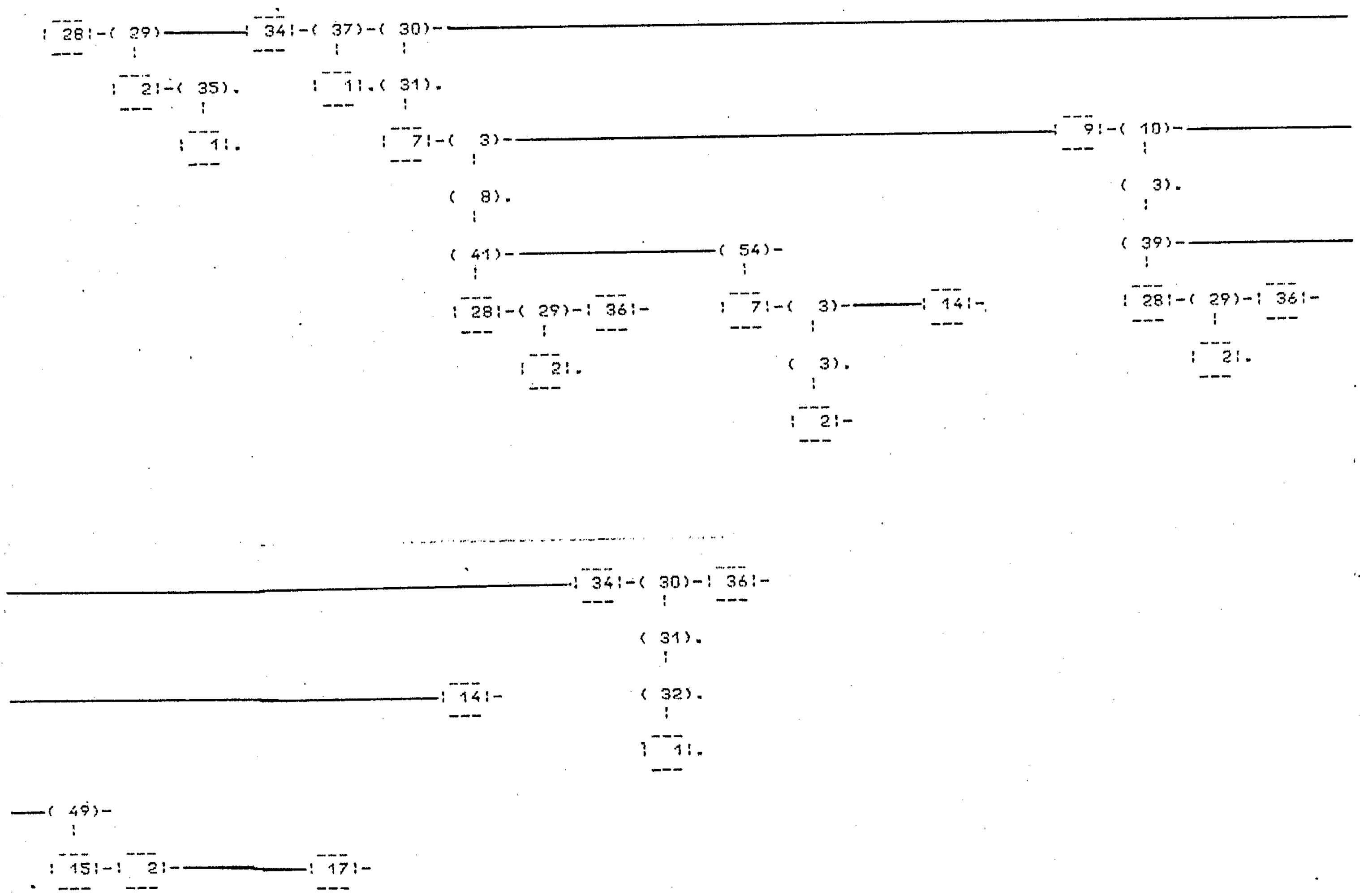
O programa ÁRVORE lê o arquivo em disco COMP gerado pelo tradutor, contendo a árvore de código, e gera uma saída impressa, contendo o desenho da árvore.

Nesta saída, os nós de código estão representados da maneira mais semelhante possível à representação definida na Seção 2 do Capítulo VI, dadas as limitações de um desenho feito por computador.

Na Figura VII.2 há um exemplo da saída do programa que desenha árvores.

Fig. VII.2

Saída do Programa que Desenha Arvores



VIII. CONCLUSÕES

A nosso ver, o principal mérito desta tese foi o progresso efetuado durante o próprio desenrolar do trabalho. Embora se tivesse começado com um conjunto de propostas para a implementação do tradutor, a medida que o trabalho ia caminhando diversas novas idéias foram sendo incorporadas, afetando fundamentalmente o resultado final.

Não se tratou simplesmente de escrever um programa, dado um método de análise sintática consagrado pelo uso e gerando um tipo de código intermediário previamente definido. Se isto fosse, o elemento criativo teria sido mínimo.

Não se tratou tampouco de uma discussão teórica sobre possíveis métodos para análise sintática e geração de código intermediário para linguagens de banco de dados.

O que se fez foi buscar, até achar, os métodos e idéias mais adequados para o fim em vista, com base na teoria e nos resultados práticos que foram sendo obtidos, e implementá-los.

O método RRP para a análise sintática não foi o primeiro a ser tentado, mas foi usado por se ter chegado à conclusão, com base na prática, de que era o mais próprio em nosso caso. Não conhecemos nenhuma outra implementação deste tipo que tenha usado este método, tratando-se de um caminho até então pouco explorado.

O uso de árvore como saída de um tradutor também é relativamente pouco usual, sendo potencialmente muito forte, na medida das possibilidades que ele fornece ao interpretador.

E, consequentemente, a implementação de um tradutor usando uma gramática RRP como entrada e gerando árvore em sua saída deve ser algo inédito, representando uma contribuição significativa para a teoria da compilação.

Contribuição significativa, acreditamos, foi dada ainda na área de compilação de linguagens de acesso a banco de dados, que, em geral, e particularmente no Brasil, é um processo artesanal,muito pouco formalizado. Por exemplo, a idéia do embutimento do código para a análise semântica no código executável é um primeiro passo na busca de métodos de tratamento formais de semântica de banco de dados.

Quando todo o conjunto tradutor/interpretador/executor for implementado, poder-se-á efetuar uma análise criteriosa da performance do sistema. De qualquer forma, o tempo gasto na análise e tradução será sempre mínimo, se comparado com o tempo gasto pela interpretação e execução do código.

Uma linha de programa fonte analisada e traduzida quase que instantaneamente pode até levar um tempo da ordem de minutos para ser executada, já que a própria definição da linguagem coloca instrumentos fortíssimos na mão do usuário. A implementação das primitivas de acesso ao banco de dados é tarefa bastante complexa, dadas as limitações do sistema portador.

O programa foi escrito em LPS, a linguagem de programação de sistemas, suportada pelo minicomputador COBRA-300. Trata-se de uma linguagem de médio nível, com sintaxe bastante parecida com a do Pascal. O autor não possuia experiência anterior com este equipamento ou linguagem, mas o aprendizado não ofereceu grandes dificuldades.

O programa foi totalmente escrito usando técnicas de programação estruturada. Particularmente útil foi o uso de diagramas de programação estruturada, antes da codificação. Nestes diagramas o programa é representado através de blocos que, por sua vez,representam os três tipos de construções permitidos: seqüência,decisão e iteração. Com o uso desta técnica praticamente assegura se a inexistência de erros de lógica no programa e obtem-se uma estrutura modular facilmente modificável. Além disso, o programa é de fácil leitura.

O programa fonte tem 905 linhas e o programa objeto gerado ocupa 22 KBYTES.

A tarefa de implementar o analisador poderia ter sido bem mais suave se outros programas auxiliares já estivessem disponíveis - no computador B-6700 da UFRJ, em complemento ao CODIFICADOR RRP e ao ALTERADOR RRP.

Todas as vezes em que a sintaxe foi modificada, e não foram poucas, houve necessidade de recalcular os conjuntos FIRST e FOLLOW de cada não terminal da gramática. Além disso, foi necessário passar a nova tabela de análise sintática, do formato matriz para o formato lista.

Estas duas tarefas, que podem ser facilmente executadas por um programa desenvolvido para este fim, foram efetuadas diversas vezes, de forma manual, pelo autor desta tese. Tratando-se de tarefas braçais e maçantes, exigiram grande quantidade de tempo para a sua execução, tempo esse que poderia ter sido usado em atividades mais criativas e que real benefício trouxessem para o projeto do tradutor.

A necessidade de se desenvolver um programa para desenhar as árvores de código não estava prevista inicialmente, mas se tornou patente a medida que o trabalho foi avançando. Apesar de ser totalmente satisfatório na forma como foi implementado, ele apresenta algumas limitações, já que não se trata do objetivo central desta tese. Se sua utilidade para o projeto Miniban vier a se comprovar, ele poderá ser facilmente modificado para que produza árvores mais elegantes, do ponto de vista estético.

Temos esperança de ver o interpretador implementado proximamente, para que o Microloban possa efetivamente ser utilizado, já que se trata de software nacional desenvolvido em máquina nacional.

IX. APÊNDICES

1. TABELA DE TRADUÇÃO

REPRESENTAÇÃO	TIPO	VALOR
";"	PONTO-E-VÍRGULA	
","	VÍRGULA	
"."	PONTO	
"="	IGUAL	
"+"	MAIS	
"/"	DIVIDE	
"{"	ABRE-CHAVES	
"}"	FECHA-CHAVES	
"["	ABRE-COLCHETES	
"]"	FECHA-COLCHETES	
"|"	MARCA	
"<"	MENOR	
">"	MAIOR	
":"	DOIS-PONTOS	
"-"	MENOS	
"*"	VEZES	
")"	FECHA-PARÊNTESES	
"("	ABRE-PARÊNTESES	
"#"	CARDINAL	
"<>"	DIFERENTE	
"≤"	MENOR-OU-IGUAL	
"≥"	MAIOR-OU-IGUAL	
":="	ATRIBUIÇÃO	
"→"	IMPLICA	
"**"	EXPONENCIAÇÃO	
")d)"	FECHA-PARÊNTESES	
"(d("	ABRE-PARÊNTESES	
l(l/d/"_")	IDENTIFICADOR	REPRESENT. EXTERNA
d^+	INTEIRO	REPRESENT. EXTERNA
d^+ "," d^+	REAL	REPRESENT. EXTERNA
dd "." dd "." $(dd)^?$dd	DATA	REPRESENT. EXTERNA
dd ":" dd $(":" dd)^?$	HORA	REPRESENT. EXTERNA
ASPAS Σ^* ASPAS	NUMCAR	REPRESENT. EXTERNA

TABELA DE TRADUÇÃO - CONTINUAÇÃO

REPRESENTAÇÃO	TIPO	VALOR
"A"	A	
"C"	C	
"E"	E	
"M"	M	
"N"	N	
"U"	U	
"AO"	AO	
"CC"	CC	
"DE"	DE	
"EM"	EM	
"OU"	OU	
"PC"	PC	
"PX"	PX	
"SE"	SE	
"ASC"	ASC	
"ATE"	ATE	
"AUX"	AUX	
"COM"	COM	
"DIF"	DIF	
"INT"	INT	
"LER"	LER	
"LIG"	LIG	
"MAX"	MAX	
"MIN"	MIN	
"NÃO"	NAO	
"POR"	POR	
"QUE"	QUE	
"REC"	REC	
"TAL"	TAL	
"TUP"	TUP	
"UNI"	UNI	
"VAZ"	VAZ	

TABELA DE TRADUÇÃO - CONTINUAÇÃO

REPRESENTAÇÃO	TIPO	VALOR
"ALIG"	ALIG	
"AREA"	AREA	
"AREL"	AREL	
"CADA"	CADA	
"CALC"	CALC	
"CARD"	CARD	
"CONT"	CONT	
"DATA"	DATA	
"DESC"	DESC	
"ELEM"	ELEM	
"EXCL"	EXCL	
"FITA"	FITA	
"HORA"	HORA	
"IMPL"	IMPL	
"INST"	INST	
"INTR"	INTR	
"JUNT"	JUNT	
"LIGA"	LIGA	
"NOME"	NOME	
"NREC"	NREC	
"OCOR"	OCOR	
"OUEX"	OUEX	
"PARA"	PARA	
"REAL"	REAL	
"TIPO"	TIPO	
"TODO"	TODO	
"TRAB"	TRAB	
"ABRIR"	ABRIR	
"ACREC"	ACREC	
"ACSET"	ACSET	
"AGRUP"	AGRUP	
"ALRG"	ALRG	
"CAMPO"	CAMPO	

TABELA DE TRADUÇÃO - CONTINUAÇÃO

REPRESENTAÇÃO	TIPO	VALOR
"CHAVE"	CHAVE	
"COLEC"	COLEC	
"CONEX"	CONEX	
"CRIAR"	CRIAR	
"DISCO"	DISCO	
"FAZER"	FAZER	
"FIXAR"	FIXAR	
"FONTE	FONTE	
"INTER"	INTER	
"LIGUE"	LIGUE	
"LIVRE"	LIVRE	
"MEDIA"	MEDIA	
"ORDEM"	ORDEM	
"RENOM"	RENOM	
"RRTUP"	RRTUP	
"SAIDA"	SAIDA	
"SENÃO"	SENÃO	
"SOBRE"	SOBRE	
"SUBST"	SUBST	
"TALIG"	TALIG	
"TAREL"	TAREL	
"TEXTO"	TEXTO	
"TIRAR"	TIRAR	
"TOTAL"	TOTAL	
"TRANS"	TRANS	
"VIDEO"	VIDEO	
"ABOLIR"	ABOLIR	
"ACTRAB"	ACTRAB	
"COMPOS"	COMPOS	
"DESVIO"	DESVIO	
"EXISTE"	EXISTE	
"FECHAR"	FECHAR	
"F_ENTR"	F_ENTR	

TABELA DE TRADUÇÃO - CONTINUAÇÃO

REPRESENTAÇÃO	TIPO	VALOR
"F_LEVE"	F_LEVE	
"F_SAID"	F_SAID	
"NUMCAR"	NUMCAR	
"PARTIR"	PARTIR	
"RRITEM"	RRITEM	
"SENHAC"	SENHAC	
"VERSÃO"	VERSÃO	
"ALTERAR"	ALTERAR	
"DESCCNT"	DESCONT	
"ENTRADA"	ENTRADA	
"ESTREIT"	ESTREIT	
"EXCLUIR"	EXCLUIR	
"F_GRAVE"	F_GRAVE	
"INCLUIR"	INCLUIR	
"PERMITE"	PERMITE	
"RRCOL_1"	RRCOL_1	
"RRCOL_2"	RRCOL_2	
"RRLIG_1"	RRLIG_1	
"RRLIG_2"	RRLIG_2	
"RRLIG_3"	RRLIG_3	
"SEGUNDO"	SEGUNDO	
"SUBCONJ"	SUBCONJ	
"TECLADO"	TECLADO	
"T_VAUTO"	T_VAUTO	
"T_VCOER"	T_VCOER	
"T_VPROT"	T_VPROT	
"USUÁRIO"	USUÁRIO	
"VALBOOL"	VALBOOL	
"VERBETE"	VERBETE	
"ARQUIVAR"	ARQUIVAR	
"COLITENS"	COLITENS	
"COMANDOS"	COMANDOS	

TABELA DE TRADUÇÃO - CONTINUAÇÃO

REPRESENTAÇÃO	TIPO	VALOR
"COMPILAR"	COMPILAR	
"DATA_ORD"	DATA_ORD	
"DESAGRUP"	DESAGRUP	
"ENCERRAR"	ENCERRAR	
"EXECUTAR"	EXECUTAR	
"HORA_ORD"	HORA_ORD	
"NUMÉRICO"	NUMÉRICO	
"PRIMÁRIA"	PRIMÁRIA	
"PROTEÇÃO"	PROTEÇÃO	
"ABANDONAR"	ABANDONAR	
"ACONTECER"	ACONTECER	
"CONSULTAR"	CONSULTAR	
"DATA_CORR"	DATA_CORR	
"GERENCIAR"	GERENCIAR	
"HORA_CORR"	HORA_CORR	
"INTERAÇÃO"	INTERAÇÃO	
"MODIFICAR"	MODIFICAR	
"RRTALIG_1"	RRTALIG_1	
"RRTALIG_2"	RRTALIG_2	
"RRTALIG_3"	RRTALIG_3	
"RRTAREL_1"	RRTAREL_1	
"RRTAREL_2"	RRTAREL_2	
"RRTAREL_3"	RRTAREL_3	
"TRANSAÇÃO"	TRANSAÇÃO	
"CARACTERES"	CARACTERES	
"COMPONENTE"	COMPONENTE	
"CONSIDERAR"	CONSIDERAR	
"DESFAZENDO"	DESFAZENDO	
"EXECUTANDO"	EXECUTANDO	
"IMPRESSORA"	IMPRESSORA	
"INSTRUÇÕES"	INSTRUÇÕES	
"SUBSTITUIR"	SUBSTITUIR	
"VERDADEIRO"	VERDADEIRO	

TABELA DE TRADUÇÃO - CONTINUAÇÃO

REPRESENTAÇÃO	TIPO	VALOR
"ESTABELECER"	ESTABELECER	
"RECONSTRUIR"	RECONSTRUIR	
"REPRESENTAR"	REPRESENTAR	
"ALFANUMÉRICO"	ALFANUMÉRICO	
"TELEIMPRESSORA"	TELEIMPRESSORA	

2. CÁLCULO DOS CONJUNTOS FIRST E FOLLOW DOS NÃO TERMINAIS

PRODUÇÃO 01

FIRST(TRABALHO) = EXECUTAR, COMPILAR

FOLLOW(COMANDO) = ;

FOLLOW(TRABALHO)= $

PRODUÇÃO 02

FIRST(COMANDO) = CRIAR, ABRIR, ABOLIR, ABANDONAR,
ARQUIVAR, RECONSTRUIR, FECHAR ,
REPRESENTAR, CONSIDERAR , FAZER ,
FIRST(TRANSA-SEM-FAZER)

FOLLOW(TRANSA-SEM-FAZER) = FOLLOW(COMANDO)

FOLLOW(EXPRESSÃO) = EM

FOLLOW(END-CAMPO) = FOLLOW(COMANDO)

FOLLOW(INST-TRANSAÇÃO) = PAR-FECHA , ;

FOLLOW(TERMO-CONTROLE) = PAR-ABRE , COM

FOLLOW(EXP-BOOLEANA) = PAR-ABRE

FOLLOW(INST-FAZER-PARA) = PAR-FECHA

PRODUÇÃO 03

FIRST(END-CAMPO) = CC , VOLUME

PRODUÇÃO 04

FIRST(END-PONTO) = ACTRAB , AUX , PC , PX , ":" ,
ABRE-COL , FIRST(EXP-BOOLEANA), ID

FOLLOW(EXP-BOOLEANA) = "." , FOLLOW(END-PONTO)

PRODUÇÃO 05

FIRST(EXP-BOOLEANA) = NÃO , FIRST(PRIM-BOOLEANO)

FOLLOW(PRIM-BOOLEANO) = E , OU , OUEX , IMPL ,
FOLLOW(EXP-BOOLEANA)

CÁLCULO DOS CONJUNTOS FIRST E FOLLOW DOS NÃO TERMINAIS-CONT.

PRODUÇÃO 06

FIRST(PRIM-BOOLEANO)	= PARA , EXISTE, C , N , VERDADEIRO. PAR-ABRE
FOLLOW(END-PONTO)	= PAR-ABRE, ELEM , '=' , '<' , '>' , '<=' , '>=' , '<>'
FOLLOW(EXP-BOOLEANA)	= PAR-FECHA
FOLLOW(TERMO-CONJUNTO)	= FOLLOW(PRIM-BOOLEANO)
FOLLOW(EXPRESSÃO)	= EM , FOLLOW(PRIM-BOOLEANO)

PRODUÇÃO 07

FIRST(EXPRESSÃO)	= FIRST(TERMO)
FOLLOW(TERMO)	= '+' , '-' , '/' , '*' , '**' , 'UNI' , 'DIF' , 'INTER', FOLLOW(EXPRESSÃO)

PRODUÇÃO 08

FIRST(TERMO)	= INTER , FIRST(CONSTANTE), N , C , COLEC , CONT, M , CALC , ALARG , CARD , DESAGRUP, ESTREIT, AGRUP , RENOM, JUNT , LIGA , VAZ , HORA-CORR , DATA-CORR , VERBETE , PAR-ABRE
FOLLOW(END-CAMPO)	= FOLLOW(TERMO)
FOLLOW(CONSTANTE)	= FOLLOW(TERMO)
FOLLOW(END-PONTO)	= FOLLOW(TERMO)
FOLLOW(EXPRESSÃO)	= DE , PAR-FECHA , FOLLOW(TERMO) , POR , SUBST , COM, EXCL, PAR-ABRE
FOLLOW(VERB)	= FOLLOW(TERMO)

CÁLCULO DOS CONJUNTOS FIRST E FOLLOW DOS NÃO TERMINAIS-CONT.

PRODUÇÃO 09

FIRST(TERMO-CONJUNTO) = ACTRAB , ALIG, AREL , COL ITENS , DATA , HORA , INT , REAL, NUMCAR, LIG , TUP , TAREL , TALIG , NOME, VALBOOL

FOLLOW(CONSTANTE) = ';' , FECHA-CHAVE

FOLLOW(VERB) = ':' , FECHA-CHAVE

FOLLOW(EXPRESSÃO = FOLLOW(TERMO-CONJUNTO)

PRODUÇÃO 10

FIRST(TERMO-CONTROLE) = FIRST(END-CAMPO),FIRST(END-PONTO)

FOLLOW(END-CAMPO) = ABRE-COL , FOLLOW(TERMO-CONTROLE)

FOLLOW(END-PONTO) = ABRE-COL , FOLLOW(TERMO-CONTROLE), EM

PRODUÇÃO 11

FIRST(VERB) = USUÁRIO , SENHAC , FONTE , ID

FOLLOW(END-PONTO-CONEX) = SUBCONJ , ',' , FOLLOW(VERB)

FOLLOW(CONSTANTE) ';', FECHA-CHAVE, E, FOLLOW(VERB)

PRODUÇÃO 12

FIRST(END-PONTO-CONEX) = N , ID , VERDADEIRO

PRODUÇÃO 13

FIRST(INST-TRANSAÇÃO) = FIRST(TRANSA-SEM-FAZER) , FAZER

FOLLOW(TRANSA-SEM-FAZER) = FOLLOW(INST-TRANSAÇÃO)

FOLLOW(TERMO-CONTROLE) = COM , PAR-ABRE

FOLLOW(INST-TRANSAÇÃO) = ';' , PAR-FECHA

FOLLOW(EXP-BOOLEANA) = PAR-ABRE

CÁLCULO DOS CONJUNTOS FIRST E FOLLOW DOS NÃO TERMINAIS-CONT.

PRODUÇÃO 14

FIRST(INST-FAZER-PARA) = FIRST(TRANSA-SEM-FAZER) , FAZER , CONSIDERAR

FOLLOW(TRANSA-SEM-FAZER) = FOLLOW(INST-FAZER-PARA)

FOLLOW(TERMO-CONTROLE) = COM , PAR-ABRE

FOLLOW(INST-FAZER-PARA) = ';' , PAR-FECHA

FOLLOW(EXP-BOOLEANA) = PAR-ABRE

FOLLOW(INST-TRANSAÇÃO) = ';' , PAR-FECHA

PRODUÇÃO 15

FIRST(TRANSA-SEM-FAZER) = INCLUIR, EXCLUIR , SUBSTITUIR , FIXAR , TIRAR

FOLLOW(END-PONTO) = FOLLOW(TRANSA-SEM-FAZER) , POR

FOLLOW(EXPRESSÃO) = FOLLOW(TRANSA-SEM-FAZER) , ':=' , EM

FOLLOW(TERMO-CONJUNTO) = EM

PRODUÇÃO 16

FIRST(CONSTANTE) = CONSTANTE-DATA , CONSTANTE-HORA , CONSTANTE-NUMCAR , NINT , NREAL , '+' , '-'

FIRST

TRABALHO	EXECUTAR , COMPILAR
COMANDO	CRIAR , ABRIR , ABOLIR , ABANDONAR , ARQUIVAR , RECONSTRUIR , FECHAR , REPRESENTAR , CONSIDERAR , FAZER , INCLUIR , EXCLUIR , SUBSTITUIR , FIXAR , TIRAR
END-CAMPO	CC , VOLUME
END-PONTO	ACTRAB , AUX , PC , PX , ':' , ABRE-COL , ID , NÃO , PARA , EXISTE , C , N , VERDADEIRO , PAR-ABRE
EXP-BOOLEANA	NÃO , PARA , EXISTE , C , N , VERDADEIRO , PAR-ABRE
PRIM-BOOLEANO	PARA , EXISTE , C , N , VERDADEIRO , PAR-ABRE
EXPRESSÃO	INTER , C , N , COLEC , CONT , M , CALC , ALARG , CARD , DESAGRUP , ESTREIT , AGRUP, RENOM , JUNT , LIGA , VAZ , HORA-CORR , DATA-CORR , VERBETE , PAR-ABRE , NINT , NREAL , '+' , '=' , CONSTANTE-DATA , CONSTANTE-HORA , CONSTANTE-NUMCAR
TERMO	FIRST(EXPRESSÃO)
TERMO-CONJUNTO	ACTRAB , ALIG , AREL , COLITENS , DATA , HORA , INT , REAL , NUMCAR , LIG , TUP , TAREL , TALIG , NOME , VALBOOL , ID , ABRE-CHAVE , FIRST(EXPRESSÃO)

FIRST - CONTINUAÇÃO

TERMO-CONTROLE	CC , VOLUME , ACTRAB , AUX , PC , PX , ':' , ABRE-COL , ID , NÃO , PARA , EXIST , C , N , VERDADEIRO , PAR-ABRE
VERB	USUÁRIO , SENHAC , FONTE , ID
END-PONTO-CONEX	N , ID , VERDADEIRO
INST-TRANSAÇÃO	INCLUIR , EXCLUIR , SUBSTITUIR , FIXAR , TIRAR , FAZER
INST-FAZER-PARA	INCLUIR , EXCLUIR , SUBSTITUIR , FIXAR , TIRAR , FAZER , CONSIDERAR
TRANSA-SEM-FAZER	INCLUIR , EXCLUIR , SUBSTITUIR , FIXAR , TIRAR
CONSTANTE	CONSTANTE-DATA , CONSTANTE-HORA , CONSTANTE-NUMCAR , NINT , NREAL , '+' , '-'

FOLLOW

TRABALHO	$
COMANDO	';'
END-CAMPO	'+' , '=' , '/' ,*' , '**' , 'UNI', 'DIF', INTER , PAR-ABRE , ELEM , '=' , '<' , '>', '<=' , '>=' , '<>' , ABRE-COL , EM , ';' , PAR-FECHA , POR , COM , E , OU , OUEX , IMPL , DE , PAR-FECHA , SUBST , EXCL , ':=' , '.'
END-PONTO	FOLLOW(END-CAMPO)
EXP-BOOLEANA	FOLLOW(END-CAMPO)
EXPRESSÃO	FOLLOW(END-CAMPO)
TERMO	FOLLOW(END-CAMPO)
TERMO-CONJUNTO	FOLLOW(END-CAMPO)
TERMO-CONTROLE	COM , PAR-ABRE
VERB	FECHA-CHAVE , FOLLOW(END-CAMPO)
END-PONTO-CONEX	SUBCONJ , ',' , FOLLOW(END-CAMPO)
INST-TRANSAÇÃO	';' , PAR-FECHA
INST-FAZER-PARA	':' , PAR-FECHA
TRANSA-SEM-FAZER	';' , PAR-FECHA
CONSTANTE	FECHA-CHAVE , FOLLOW(END-CAMPO)

3. SAÍDA DO CODIFICADOR - GRAMÁTICA EM FORMA DE EXPRESSÃO REGULAR

```
TAMANHO DO ALFABETO : 66
NUMERO DE TERMINAIS : 199
NUMERO DE NAO TERMINAIS : 16
NUMERO DE PRODUCOES : 16

A L F A B E T O :

ABCDEFGHIJKLMNOPQRSTUVXWYZ1234567890:-?@[;\],./!"#$%&'()0*=?_+~<>?
```

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NUCLEO DE COMPUTAÇÃO ELETRÔNICA

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NUCLEO DE COMPUTAÇÃO ELETRÔNICA

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NUCLEO DE COMPUTAÇÃO ELETRÔNICA

MODULO CODIFICADOR DE EXPRESSOES REGULARES — FOLHA : 1

LN	PR	IMAGEM DO CARTAO	NAO TERMINAIS NOME	NAO TERMINAIS COD.	TERMINAIS NOME	TERMINAIS COD.
1	1	TRABALHO = ('EXECUTAR'/'COMPILAR') 'PARA' 'USUARIO' 'ID'	TRABALHO	200	EXECUTAR	1
					COMPILAR	2
					PARA	3
					USUARIO	4
					ID	5
2	1	'PAR-ABRE' 'ID' 'PAR-FECHA'				
					PAR-ABRE	6
					PAR-FECHA	7
3	1	('ID'/'INSTRUCOES') ':' (COMANDO ';')* 'ENCERRAR' ;				
					INSTRUCOES	8
			COMANDO	201	:	9
					;	10
					ENCERRAR	11
4	2	COMANDO = ('CRIAR' 'ACSET' 'ID' ('A' 'PARTIR' 'DE' 'ACSET' 'ID'				
					CRIAR	12
					ACSET	13
					A	14
					PARTIR	15
					DE	16
5	2	'ATE' 'VERSAO' 'NINT')? /				
					ATE	17
					VERSAO	18
					NINT	19
6	2	'ABRIR' 'ACSET' 'ID' ('COM' 'ID' & ',')?)';'				
					ABRIR	20
					COM	21
					,	22
7	2	('ABRIR' 'VOLUME' 'DE' ('ENTRADA'/'SAIDA'/'INTERACAO')				
					VOLUME	23
					ENTRADA	24
					SAIDA	25
					INTERACAO	26
8	2	'ID' ('TECLADO'/'VIDEO'/'IMPRESSORA'/'FITA'/'DISCO'/				
					TECLADO	27
					VIDEO	28
					IMPRESSORA	29
					FITA	30
					DISCO	31
9	2	'TELEIMPRESSORA') ';')*				
					TELEIMPRESSO	32
10	2	'ESTABELECER' 'PROTECAO' 'ID' ('PARA' ('LER' /'ALTERAR')				
					ESTABELECER	33
					PROTECAO	34
					LER	35
					ALTERAR	36
11	2	'SOBRE' ('ID'/'ACTRAB') & 'U') & 'E'				
					SOBRE	37

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NUCLEO DE COMPUTAÇÃO ELETRONICA

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NUCLEO DE COMPUTAÇÃO ELETRONICA

```
                                                                                SOBRE        37
                                                                                ACTRAB       38
                                                                                U            39
                                                                                E            40
12  2          / TRANSA-SEM-FAZER
                                                                TRANSA-SEM-F 202
13  2          / ('CRIAR'/'ABOLTR') ('AREL'/'ALIG') 'ID'
                                                                                ABOLIR       41
                                                                                AREL         42
                                                                                ALIG         43
14  2            / 'ABANDONAR' 'PROTECAO' 'ID'
                                                                                ABANDONAR    44
15  2             / 'ARQUIVAR' 'ACTRAB'
                                                                                ARQUIVAR     45
16  2          / 'ABANDONAR' 'ACREC'
                                                                                ACREC        46
17  2            / 'RECONSTRUIR' 'ACSET' ('EXECUTANDO'/'DESFAZENDO')
                                                                                RECONSTRUIR  47
                                                                                EXECUTANDO   48
                                                                                DESFAZENDO   49
18  2                    'NINT' 'COMANDOS'
                                                                                COMANDOS     50
19  2            / 'FECHAR' 'ACSET' ( 'ID')?
                                                                                FECHAR       51
20  2          / 'REPRESENTAR' 'SEGUNDO'
                                                                                REPRESENTAR  52
                                                                                SEGUNDO      53
21  2               ('RRITEM' /'RRTUP' / 'RRLIG-1'/'RRLIG-2'/'RRLIG-3'/
                                                                                RRITEM       54
                                                                                RRTUP        55
                                                                                RRLIG-1      56
                                                                                RRLIG-2      57
                                                                                RRLIG-3      58
22  2                  'RRTAREL-1'/'RRTAREL-2'/'RRTAREL-3'/'RRTALIG-1'/
                                                                                RRTAREL-1    59
                                                                                RRTAREL-2    60
                                                                                RRTAREL-3    61
                                                                                RRTALIG-1    62
23  2                 'RRTALIG-2'/'RRTALIG-3'/'RRCOL-1'/'RRCOL-2')
                                                                                RRTALIG-2    63
                                                                                RRTALIG-3    64
                                                                                RRCOL-1      65
                                                                                RRCOL-2      66
24  2                 EXPRESSAO 'EM' ('AREA' 'LIVRE')? END-CAMPO
                                                                EXPRESSAO    203
                                                                                EM           67
                                                                                AREA         68
                                                                END-CAMPO    204 LIVRE        69
25  2          / 'FECHAR' 'VOLUME' 'DE' ( 'ENTRADA'/'SAIDA'/'INTERACAO') 'ID'
26  2          / 'ABOLIR' 'ACSET' 'ID'
27  2            / 'CONSIDERAR' 'TRANSACAO' 'PAR-ABRE' ( 'AO' 'ACONTECER' (
                                                                                CONSIDERAR   70
                                                                                TRANSACAO    71
                                                                                AO           72
```

```
28  2          ('T-VCDER'/'T-VAUTD'/'T-VPRDT'/'F-GRAVE'/'F-LEVE'/'F-ENTR'

29  2           /'F-SAID')

30  2        ':' ( 'CDNT' 'NREC' / 'DESCDNT' ( 'TRANS'/'TRAB'/'INST')

31  2          ('REC'/'NREC')))& ',' )? (INST-TRANSACAO & ';')?
```

```
TRANSACAO    71
AD           72
ACONTECER    73

T-VCDER      74
T-VAUTD      75
T-VPRDT      76
F-GRAVE      77
F-LEVE       78
F-ENTR       79

F-SAID       80

CDNT         81
NREC         82
DESCDNT      83
TRANS        84
TRAB         85
INST         86
```

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NUCLEO DE COMPUTAÇÃO ELETRÔNICA

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NUCLEO DE COMPUTAÇÃO ELETRÔNICA

MODULO CODIFICADOR DE EXPRESSOES REGULARES — FOLHA : 2

LN	PR	IMAGEM DO CARTAO	NAO TERMINAIS NOME	COD.	TERMINAIS NOME	COD.
			INST-TRANSAC	205	REC	87
32	2	'PAR-FECHA'				
33	2	/ 'FAZER' 'PARA' TERMO-CONTROLE ('COM' TERMO-CONTROLE)?				
			TERMO-CONTRO	206	FAZER	88
34	2	'PAR-ABRE' INST-FAZER-PARA & ';' 'PAR-FECHA'				
			INST-FAZER-P	207		
35	2	/ 'FAZER' 'SE' EXP-BOOLEANA 'PAR-ABRE' INST-FAZER-PARA				
					SE	89
			EXP-BOOLEANA	208		
36	2	& ';' 'PAR-FECHA' 'SENAO' 'PAR-ABRE'				
					SENAO	90
37	2	INST-FAZER-PARA & ';' 'PAR-FECHA' ;				
38	3	END-CAMPO = 'CC' ('1' 'ID') ?				
					CC	91
					1	92
39	3	/ 'VOLUME' ('ID') ? '.' 'CAMPO' 'PAR-ABRE' 'PAR-FECHA' ;				
					.	93
					CAMPO	94
40	4	END-PONTO = (('ACTRAB'/'AUX'/('PC'/'PX') ('1' 'ID') ? / ':' 'ID') '.')				
			END-PONTO	209		
					AUX	95
					PC	96
					PX	97
41	4	? (('[' '=' 'ID' ']') ?				
					[	98
					=	99
					]	100
42	4	(EXP-BOOLEANA / 'ID')) & '.'				
43	4	/ ('ACTRAB' / 'AUX' / ('PC'/'PX') ('1' 'ID') ? / ':'				
44	4	'ID') ;				
45	5	EXP-BOOLEANA = (('NAO')? PRIM-BOOLEANO) & ('E'/'OU'/'OUEX'/'IMPL');				
			PRIM-BOOLEAN	210	NAO	101
					OU	102
					OUEX	103
					IMPL	104
46	6	PRIM-BOOLEANO = 'PARA' 'TODO' END-PONTO 'PAR-ABRE'				
					TODO	105
47	6	EXP-BOOLEANA 'PAR-FECHA'				
48	6	/ 'EXISTE' END-PONTO ('PAR-ABRE'				
					EXISTE	106
49	6	EXP-BOOLEANA ? 'PAR-FECHA')				

```
50  6                 / 'C' END-PONTO 'ELEM' TERMO CONJUNTO
                                                                                                  C          107
                                                                                TERMO-CONJUN 211  ELEM       108
51  6          / ('C'/'N') END-PONTO ('='/'<'/'>'/'<='/'>='/'<>') (EXPRESSAO) /
                                                                                                  N          109
                                                                                                  <          110
                                                                                                  >          111
                                                                                                  <=         112
                                                                                                  >=         113
                                                                                                  <>         114
52  6                    'ID') ('EM' 'ORDEM' ('NUMERICO'/'ALFANUMERICO'/
                                                                                                  ORDEM      115
                                                                                                  NUMERICO   116
                                                                                                  ALFANUMERICO 117
53  6                              'DATA-ORD'/'HORA-ORD') ) ?
                                                                                                  DATA-ORD   118
                                                                                                  HORA-ORD   119
54  6                / 'VERDADEIRO'
                                                                                                  VERDADEIRO 120
55  6              / 'PAR-ABRE' EXP-BOOLEANA 'PAR-FECHA' ;
56  7   EXPRESSAO = TERMO & ('+'/'-'/'/'/'*'/'**'/'UNI'/'DIF'/'INTER');
                                                                                TERMO        212
                                                                                                  +          121
                                                                                                  -          122
                                                                                                  /          123
                                                                                                  *          124
                                                                                                  **         125
                                                                                                  UNI        126
                                                                                                  DIF        127
                                                                                                  INTER      128
57  8   TERMO = 'INTR' 'SEGUNDO' 'ID' END-CAMPO / CONSTANTE
                                                                                CONSTANTE    213  INTR       129
58  8          / ('N'/'C'/'COLEC'/'CONT'/'M') END-PONTO
                                                                                                  COLEC      130
                                                                                                  M          131
59  8          / 'CALC' ('TOTAL'/'MEDIA'/'MAX'/'MIN'/'DESVIO') 'SOBRE'
                                                                                                  CALC       132
                                                                                                  TOTAL      133
                                                                                                  MEDIA      134
                                                                                                  MAX        135
                                                                                                  MIN        136
                                                                                                  DESVIO     137
60  8                END-PONTO
61  8          / 'ALARG' EXPRESSAO 'DE' ('ID' ('.' 'ID')?
                                                                                                  ALARG      138
```

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NUCLEO DE COMPUTAÇÃO ELETRÔNICA

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NUCLEO DE COMPUTAÇÃO ELETRÔNICA

MODULO CODIFICADOR DE EXPRESSOES REGULARES — FOLHA : 3

LN	PR	IMAGEM DO CARTAO	NAO TERMINAIS NOME	NAO TERMINAIS COD.	TERMINAIS NOME	TERMINAIS COD.
62	8	'PAR-ABRE' EXPRESSAO 'PAR-FECHA') & ','				
63	8	/ ('CARD'/'DESAGRUP') EXPRESSAO				
					CARD	139
					DESAGRUP	140
64	8	/ ('ESTREIT' EXPRESSAO 'DE' / 'AGRUP' EXPRESSAO 'POR')				
					ESTREIT	141
					AGRUP	142
					POR	143
65	8	('ID' ('.' 'ID')?)& ','				
66	8	/ 'RENOM' EXPRESSAO 'SUBST' ('ID' 'POR' 'ID') & ','				
					RENOM	144
					SUBST	145
67	8	/ ('JUNT'/'LIGA') EXPRESSAO 'COM' EXPRESSAO ('EXCL')?				
					JUNT	146
					LIGA	147
					EXCL	148
68	8	'PAR-ABRE' ('C' 'ID' ('.' 'ID') ? ('=' / '<' / '>' /				
69	8	'<=' / '>=' / '<>') 'C' 'ID' ('.' 'ID') ?) ?				
70	8	'PAR-FECHA'				
71	8	/ 'VAZ' / 'HORA-CORR' / 'DATA-CORR'				
					VAZ	149
					HORA-CORR	150
					DATA-CORR	151
72	8	/ 'VERBETE' VERB				
			VERB	214	VERBETE	152
73	8	/ 'PAR-ABRE' EXPRESSAO 'PAR-FECHA' ;				
74	9	TERMO-CONJUNTO = 'ID' / 'AC' CONSTANTE & ';' 'FC' -				
					AC	153
					FC	154
75	9	/ EXPRESSAO / 'AC' ('VERBETE' VERB) & ';' 'FC'				
76	9	/'ACTRAB'/'ALIG'/'AREL'				
77	9	/'COLITENS'/'DATA'/'HORA'/'INT'/'REAL'/'NUMCAR'/'LIG'				
					COLITENS	155
					DATA	156
					HORA	157
					INT	158
					REAL	159
					NUMCAR	160
					LIG	161
78	9	/'TUP'/'TAREL'/'TALIG'/'NOME'/'VALBOOL' ;				
					TUP	162

```
79  10  TERMO-CONTROLE = END-CAMPO ( '[' '=' 'ID' ']' )?
80  10                 / END-PONTO ( '[' '='
81  10                 'ID' ']' )? ( 'EM' 'ORDEM' 'POR'
82  10                   ( 'C' 'ID' ('.' 'ID')? ('ASC'/'DESC')) & ',' )? ;
83  11  VERB = 'USUARIO' 'ID' 'PAR-ABRE' 'ID' 'PAR-FECHA' 'COM' 'ID' & ','
84  11       / 'SENHAC' 'ID' ('PERMITE' ('GERENCIAR'/'MODIFICAR'/'CONSULTAR')
85  11                        ('ACTRAB'/'ID') ) & ','
86  11       / 'FONTE' 'ID' ':' 'TEXTO' '#'
87  11       / 'ID' ':' 'TIPO' 'DE'
88  11                  ('ACTRAB'/'ALIG'/'AREL'/
89  11                 'COLITENS'/'DATA'/'HORA'/'INT'/'REAL'/'NUMCAR'/
90  11                 'LIG'/'TUP'/'TAREL'/'TALIG'/'NOME'/'VALBOOL')
91  11          'TAL' 'QUE'
92  11           ('COMPOS' ('ID' / ('ID' '->'
```

TUP	162
TAREL	163
TALIG	164
NOME	165
VALBOOL	166
ASC	167
DESC	168
SENHAC	169
PERMITE	170
GERENCIAR	171
MODIFICAR	172
CONSULTAR	173
FONTE	174
TEXTO	175
#	176
TIPO	177
TAL	178
QUE	179
COMPOS	180

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NUCLEO DE COMPUTAÇÃO ELETRÔNICA

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NUCLEO DE COMPUTAÇÃO ELETRONICA


```
MODULO CODIFICADOR DE EXPRESSOES REGULARES                                                    FOLHA : 4

LN PR                                IMAGEM DO CARTAO                          NAO TERMINAIS        TERMINAIS
                                                                                 NOME   COD.        NOME   COD.
                                                                                                 =>          181
93  11                    ('ID'/'DATA'/'HORA'/'INT'/'REAL')) & ','
94  11                    / 'POR' 'NINT' 'CARACTERES')
                                                                                                  CARACTERES  182
95  11          ('CONEX' (('COLEC' END-PONTO-CONEX 'SUBCONJ' 'COLEC'
                                                                              END-PONTO-CO 215    CONEX       183
                                                                                                  SUBCONJ     184
96  11                      END-PONTO-CONEX ) & ','
97  11                    / 'CHAVE' 'PRIMARIA' 'ID' 'E' 'CARD' 'C'
                                                                                                  CHAVE       185
                                                                                                  PRIMARIA    186
98  11                      'OCOR' '<=' 'NINT' ))?
                                                                                                  OCOR        187
99  11          / 'C' 'OCOR' ( 'ELEM' 'AO' CONSTANTE & ';' 'FO'
100 11                      / ('='/'<'/'>'/'<='/'>='/'<>') CONSTANTE ( 'E' 'C'
101 11                      'OCOR' ('='/'<'/'>'/'<='/'>='/'<>') CONSTANTE ))));
102 12   END-PONTO-CONEX = (('N' 'PX' '=')? 'ID' / 'VERDADEIRO') & '.' ;
103 13  INST-TRANSACAO = TRANSA-SEM-FAZER
104 13                / 'FAZER' 'PARA' TERMO-CONTROLE ('COM' TERMO-CONTROLE)?
105 13                    'PAR-ABRE' INST-TRANSACAO & ';' 'PAR-FECHA'
106 13                / 'FAZER' 'SE' EXP-BOOLEANA 'PAR-ABRE' INST-TRANSACAO
107 13                        & ';' 'PAR-FECHA' 'SENAO' 'PAR-ABRE'
108 13                                  INST-TRANSACAO & ';' 'PAR-FECHA' ;
109 14  INST-FAZER-PARA = TRANSA-SEM-FAZER
110 14                / 'FAZER' 'PARA' TERMO-CONTROLE ('COM' TERMO-CONTROLE)?
111 14                    'PAR-ABRE' INST-FAZER-PARA & ';' 'PAR-FECHA'
112 14                / 'FAZER' 'SE' EXP-BOOLEANA 'PAR-ABRE' INST-FAZER-PARA
113 14                        & ';' 'PAR-FECHA' 'SENAO' 'PAR-ABRE'
114 14                                  INST-FAZER-PARA & ';' 'PAR-FECHA'
115 14                / 'CONSIDERAR' 'TRANSACAO' 'PAR-ABRE' ( 'AO'
116 14                    'ACONTECER' ( ('F-VCOER'/'F-VAUTO'/'F-VPROT'/
117 14                    'F-GRAVE'/'F-LEVE'/'F-ENTR'/'F-SAID') ';'
```

```
118 14                    ( 'CONT' 'NREC'  / 'DESCONT' ('TRANS'/'TRAB'/
119 14                    'INST') ('REC'/'NREC'))) & ',' )?
120 14                    (INST-TRANSACAO & ';')? 'PAR-FECHA' ;
121 15  TRANSA-SEM-FAZER = 'INCLUIR' ( EXPRESSAO (':=' EXPRESSAO )? / 'CADA'
                                                                    INCLUIR     188
                                                                    :=          189
                                                                    CADA        190
122 15                    'COMPONENTE' 'DE' TERMO-CONJUNTO ) 'EM' END-PONTO
                                                                    COMPONENTE  191
```

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NUCLEO DE COMPUTAÇÃO ELETRONICA

```
==================================================================================================
                MODULO CODIFICADOR DE EXPRESSOES REGULARES                          FOLHA : 5
==================================================================================================
LN PR                          IMAGEM DO CARTAO                          NAO TERMINAIS      TERMINAIS
                                                                          NOME    COD.       NOME    COD.
123 15              / 'EXCLUIR' 'DE' END-PONTO
                                                                                             EXCLUIR      192
124 15              / 'SUBSTITUIR' 'EM' END-PONTO 'POR' EXPRESSAO
                                                                                             SUBSTITUIR   193
125 15              / 'FIXAR' 'ID' 'EM' END-PONTO
                                                                                             FIXAR        194
126 15              / 'TIRAR' 'ID' & ',' ('DE' END-PONTO)? ;
                                                                                             TIRAR        195
127 16  CONSTANTE =  'CONSTANTE-DATA' /
                                                                                             CONSTANTE-DA 196
128 16              'CONSTANTE-HORA' / 'CONSTANTE-NUMCAR' /
                                                                                             CONSTANTE-HO 197
                                                                                             CONSTANTE-NU 198
129 16          (('+')?/'-') 'NINT' / (('+')?/'-') 'NREAL' ;
                                                                                             NREAL        199
=========================================================
FINAL DO CODIFICADOR
=========================================================
                NUMERO DE PRODUCOES CODIFICADAS :    16
                NUMERO DE TERMINAIS FORMADOS :   199
                NUMERO DE NAO TERMINAIS FORMADOS :    16
                NUMERO DE NOS CRIADOS :  1426

GRAVACAO DO ARQUIVO COMPLETADA NORMALMENTE
```

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NUCLEO DE COMPUTAÇÃO ELETRÔNICA

4. SAÍDA DO ALTERADOR - GRAMÁTICA EM FORMA DE AFD

```
AUTOMATO PRODUCAO #    1
  ESTADO :    1
    SIMBOLO    1 - TRANSICAO    2
    SIMBOLO    2 - TRANSICAO    2
  ESTADO :    2
    SIMBOLO    3 - TRANSICAO    3
  ESTADO :    3
    SIMBOLO    4 - TRANSICAO    4
  ESTADO :    4
    SIMBOLO    5 - TRANSICAO    5
  ESTADO :    5
    SIMBOLO    6 - TRANSICAO    6
  ESTADO :    6
    SIMBOLO    5 - TRANSICAO    7
  ESTADO :    7
    SIMBOLO    7 - TRANSICAO    8
  ESTADO :    8
    SIMBOLO    5 - TRANSICAO    9
    SIMBOLO    8 - TRANSICAO    9
  ESTADO :    9
    SIMBOLO    9 - TRANSICAO   10
  ESTADO :   10
    SIMBOLO   11 - TRANSICAO   11
    SIMBOLO  201 - TRANSICAO   12
  ESTADO :   11F
  ESTADO :   12
    SIMBOLO   10 - TRANSICAO   10
```

```
AUTOMATO MINIMO
  ESTADO :    1
    SIMBOLO    1 - TRANSICAO    2
    SIMBOLO    2 - TRANSICAO    2
  ESTADO :    2
    SIMBOLO    3 - TRANSICAO    3
  ESTADO :    3
    SIMBOLO    4 - TRANSICAO    4
  ESTADO :    4
    SIMBOLO    5 - TRANSICAO    5
  ESTADO :    5
    SIMBOLO    6 - TRANSICAO    6
  ESTADO :    6
    SIMBOLO    5 - TRANSICAO    7
  ESTADO :    7
    SIMBOLO    7 - TRANSICAO    8
  ESTADO :    8
    SIMBOLO    5 - TRANSICAO    9
    SIMBOLO    8 - TRANSICAO    9
  ESTADO :    9
    SIMBOLO    9 - TRANSICAO   10
  ESTADO :   10
    SIMBOLO   11 - TRANSICAO   11
    SIMBOLO  201 - TRANSICAO   12
  ESTADO :   11F
  ESTADO :   12
    SIMBOLO   10 - TRANSICAO   10
```

AUTOMATO PRODUCAO # 2

ESTADO :	13		
SIMBOLO	12	- TRANSICAO	22
SIMBOLO	20	- TRANSICAO	24
SIMBOLO	41	- TRANSICAO	16
SIMBOLO	44	- TRANSICAO	20
SIMBOLO	45	- TRANSICAO	21
SIMBOLO	47	- TRANSICAO	19
SIMBOLO	51	- TRANSICAO	17
SIMBOLO	52	- TRANSICAO	18
SIMBOLO	70	- TRANSICAO	15
SIMBOLO	88	- TRANSICAO	14
SIMBOLO	202	- TRANSICAO	23
ESTADO :	14		
SIMBOLO	3	- TRANSICAO	26
SIMBOLO	89	- TRANSICAO	25
ESTADO :	15		
SIMBOLO	71	- TRANSICAO	27
ESTADO :	16		
SIMBOLO	13	- TRANSICAO	28
SIMBOLO	42	- TRANSICAO	29
SIMBOLO	43	- TRANSICAO	29
ESTADO :	17		
SIMBOLO	13	- TRANSICAO	31
SIMBOLO	23	- TRANSICAO	30
ESTADO :	18		
SIMBOLO	53	- TRANSICAO	32
ESTADO :	19		
SIMBOLO	13	- TRANSICAO	33
ESTADO :	20		
SIMBOLO	34	- TRANSICAO	34
SIMBOLO	46	- TRANSICAO	23
ESTADO :	21		
SIMBOLO	38	- TRANSICAO	23
ESTADO :	22		
SIMBOLO	13	- TRANSICAO	35
SIMBOLO	42	- TRANSICAO	29
SIMBOLO	43	- TRANSICAO	29
ESTADO :	23F		
ESTADO :	24		
SIMBOLO	13	- TRANSICAO	36
ESTADO :	25		
SIMBOLO	208	- TRANSICAO	37
ESTADO :	26		
SIMBOLO	206	- TRANSICAO	38
ESTADO :	27		
SIMBOLO	6	- TRANSICAO	39
ESTADO :	28		
SIMBOLO	5	- TRANSICAO	23
ESTADO :	29		
SIMBOLO	5	- TRANSICAO	23
ESTADO :	30		
SIMBOLO	16	- TRANSICAO	40
ESTADO :	31F		
SIMBOLO	5	- TRANSICAO	23

AUTOMATO MINIMO

ESTADO :	13		
SIMBOLO	12	- TRANSICAO	22
SIMBOLO	20	- TRANSICAO	24
SIMBOLO	41	- TRANSICAO	16
SIMBOLO	44	- TRANSICAO	20
SIMBOLO	45	- TRANSICAO	21
SIMBOLO	47	- TRANSICAO	19
SIMBOLO	51	- TRANSICAO	17
SIMBOLO	52	- TRANSICAO	18
SIMBOLO	70	- TRANSICAO	15
SIMBOLO	88	- TRANSICAO	14
SIMBOLO	202	- TRANSICAO	23
ESTADO :	14		
SIMBOLO	3	- TRANSICAO	26
SIMBOLO	89	- TRANSICAO	25
ESTADO :	15		
SIMBOLO	71	- TRANSICAO	27
ESTADO :	16		
SIMBOLO	13	- TRANSICAO	28
SIMBOLO	42	- TRANSICAO	28
SIMBOLO	43	- TRANSICAO	28
ESTADO :	17		
SIMBOLO	13	- TRANSICAO	30
SIMBOLO	23	- TRANSICAO	29
ESTADO :	18		
SIMBOLO	53	- TRANSICAO	31
ESTADO :	19		
SIMBOLO	13	- TRANSICAO	32
ESTADO :	20		
SIMBOLO	34	- TRANSICAO	28
SIMBOLO	46	- TRANSICAO	23
ESTADO :	21		
SIMBOLO	38	- TRANSICAO	23
ESTADO :	22		
SIMBOLO	13	- TRANSICAO	33
SIMBOLO	42	- TRANSICAO	28
SIMBOLO	43	- TRANSICAO	28
ESTADO :	23F		
ESTADO :	24		
SIMBOLO	13	- TRANSICAO	34
ESTADO :	25		
SIMBOLO	208	- TRANSICAO	35
ESTADO :	26		
SIMBOLO	206	- TRANSICAO	36
ESTADO :	27		
SIMBOLO	6	- TRANSICAO	37
ESTADO :	28		
SIMBOLO	5	- TRANSICAO	23
ESTADO :	29		
SIMBOLO	16	- TRANSICAO	38
ESTADO :	30F		
SIMBOLO	5	- TRANSICAO	23
ESTADO :	31		
SIMBOLO	54	- TRANSICAO	39
SIMBOLO	55	- TRANSICAO	39

	SIMBOLO 56 - TRANSICAO 39
	SIMBOLO 57 - TRANSICAO 39
	SIMBOLO 58 - TRANSICAO 39
	SIMBOLO 59 - TRANSICAO 39
	SIMBOLO 60 - TRANSICAO 39
	SIMBOLO 61 - TRANSICAO 39
	SIMBOLO 62 - TRANSICAO 39
	SIMBOLO 63 - TRANSICAO 39
	SIMBOLO 64 - TRANSICAO 39
	SIMBOLO 65 - TRANSICAO 39
	SIMBOLO 66 - TRANSICAO 39
ESTADO : 32	ESTADO : 32
SIMBOLO 54 - TRANSICAO 41	SIMBOLO 48 - TRANSICAO 40
SIMBOLO 55 - TRANSICAO 41	SIMBOLO 49 - TRANSICAO 40
SIMBOLO 56 - TRANSICAO 41	
SIMBOLO 57 - TRANSICAO 41	
SIMBOLO 58 - TRANSICAO 41	
SIMBOLO 59 - TRANSICAO 41	
SIMBOLO 60 - TRANSICAO 41	
SIMBOLO 61 - TRANSICAO 41	
SIMBOLO 62 - TRANSICAO 41	
SIMBOLO 63 - TRANSICAO 41	
SIMBOLO 64 - TRANSICAO 41	
SIMBOLO 65 - TRANSICAO 41	
SIMBOLO 66 - TRANSICAO 41	
ESTADO : 33	ESTADO : 33
SIMBOLO 48 - TRANSICAO 42	SIMBOLO 5 - TRANSICAO 41
SIMBOLO 49 - TRANSICAO 42	
ESTADO : 34	ESTADO : 34
SIMBOLO 5 - TRANSICAO 23	SIMBOLO 5 - TRANSICAO 42
ESTADO : 35	ESTADO : 35
SIMBOLO 5 - TRANSICAO 43	SIMBOLO 6 - TRANSICAO 43
ESTADO : 36	ESTADO : 36
SIMBOLO 5 - TRANSICAO 44	SIMBOLO 6 - TRANSICAO 44
	SIMBOLO 21 - TRANSICAO 45
ESTADO : 37	ESTADO : 37
SIMBOLO 6 - TRANSICAO 45	SIMBOLO 7 - TRANSICAO 23
	SIMBOLO 72 - TRANSICAO 47
	SIMBOLO 205 - TRANSICAO 46
ESTADO : 38	ESTADO : 38
SIMBOLO 6 - TRANSICAO 46	SIMBOLO 24 - TRANSICAO 28
SIMBOLO 21 - TRANSICAO 47	SIMBOLO 25 - TRANSICAO 28
	SIMBOLO 26 - TRANSICAO 28
ESTADO : 39	ESTADO : 39
SIMBOLO 7 - TRANSICAO 23	SIMBOLO 203 - TRANSICAO 48
SIMBOLO 72 - TRANSICAO 49	
SIMBOLO 205 - TRANSICAO 48	
ESTADO : 40	ESTADO : 40
SIMBOLO 24 - TRANSICAO 50	SIMBOLO 19 - TRANSICAO 49
SIMBOLO 25 - TRANSICAO 50	
SIMBOLO 26 - TRANSICAO 50	
ESTADO : 41	ESTADO : 41
SIMBOLO 203 - TRANSICAO 51	SIMBOLO 10 - TRANSICAO 50
	SIMBOLO 14 - TRANSICAO 51
ESTADO : 42	ESTADO : 42
SIMBOLO 19 - TRANSICAO 52	SIMBOLO 10 - TRANSICAO 50

ESTADO : 43
SIMBOLO 10 - TRANSICAO 53
SIMBOLO 14 - TRANSICAO 54

ESTADO : 44
SIMBOLO 10 - TRANSICAO 53
SIMBOLO 21 - TRANSICAO 55

ESTADO : 45
SIMBOLO 207 - TRANSICAO 56

ESTADO : 46
SIMBOLO 207 - TRANSICAO 57

ESTADO : 47
SIMBOLO 206 - TRANSICAO 58

ESTADO : 48
SIMBOLO 7 - TRANSICAO 23
SIMBOLO 10 - TRANSICAO 59

ESTADO : 49
SIMBOLO 73 - TRANSICAO 60

ESTADO : 50
SIMBOLO 5 - TRANSICAO 23

ESTADO : 51
SIMBOLO 67 - TRANSICAO 61

ESTADO : 52
SIMBOLO 50 - TRANSICAO 23

ESTADO : 53
SIMBOLO 20 - TRANSICAO 63
SIMBOLO 33 - TRANSICAO 62

ESTADO : 54
SIMBOLO 15 - TRANSICAO 64

ESTADO : 55
SIMBOLO 5 - TRANSICAO 65

ESTADO : 56
SIMBOLO 7 - TRANSICAO 66
SIMBOLO 10 - TRANSICAO 67

ESTADO : 57
SIMBOLO 7 - TRANSICAO 23
SIMBOLO 10 - TRANSICAO 68

ESTADO : 58
SIMBOLO 6 - TRANSICAO 46

ESTADO : 59
SIMBOLO 205 - TRANSICAO 48

ESTADO : 60
SIMBOLO 74 - TRANSICAO 69
SIMBOLO 75 - TRANSICAO 69
SIMBOLO 76 - TRANSICAO 69
SIMBOLO 77 - TRANSICAO 69
SIMBOLO 78 - TRANSICAO 69

SIMBOLO 21 - TRANSICAO 52

ESTADO : 43
SIMBOLO 207 - TRANSICAO 53

ESTADO : 44
SIMBOLO 207 - TRANSICAO 54

ESTADO : 45
SIMBOLO 206 - TRANSICAO 55

ESTADO : 46
SIMBOLO 7 - TRANSICAO 23
SIMBOLO 10 - TRANSICAO 56

ESTADO : 47
SIMBOLO 73 - TRANSICAO 57

ESTADO : 48
SIMBOLO 67 - TRANSICAO 58

ESTADO : 49
SIMBOLO 50 - TRANSICAO 23

ESTADO : 50
SIMBOLO 20 - TRANSICAO 60
SIMBOLO 33 - TRANSICAO 59

ESTADO : 51
SIMBOLO 15 - TRANSICAO 61

ESTADO : 52
SIMBOLO 5 - TRANSICAO 62

ESTADO : 53
SIMBOLO 7 - TRANSICAO 63
SIMBOLO 10 - TRANSICAO 43

ESTADO : 54
SIMBOLO 7 - TRANSICAO 23
SIMBOLO 10 - TRANSICAO 44

ESTADO : 55
SIMBOLO 6 - TRANSICAO 44

ESTADO : 56
SIMBOLO 205 - TRANSICAO 46

ESTADO : 57
SIMBOLO 74 - TRANSICAO 64
SIMBOLO 75 - TRANSICAO 64
SIMBOLO 76 - TRANSICAO 64
SIMBOLO 77 - TRANSICAO 64
SIMBOLO 78 - TRANSICAO 64
SIMBOLO 79 - TRANSICAO 64
SIMBOLO 80 - TRANSICAO 64

ESTADO : 58
SIMBOLO 68 - TRANSICAO 65
SIMBOLO 204 - TRANSICAO 23

ESTADO : 59
SIMBOLO 34 - TRANSICAO 66

ESTADO : 60
SIMBOLO 23 - TRANSICAO 67

SIMBOLO 79 - TRANSICAO 69	
SIMBOLO 80 - TRANSICAO 69	
ESTADO : 61	ESTADO : 61
SIMBOLO 68 - TRANSICAO 70	SIMBOLO 16 - TRANSICAO 68
SIMBOLO 204 - TRANSICAO 23	
ESTADO : 62	ESTADO : 62
SIMBOLO 34 - TRANSICAO 71	SIMBOLO 10 - TRANSICAO 50
	SIMBOLO 22 - TRANSICAO 52
ESTADO : 63	ESTADO : 63
SIMBOLO 23 - TRANSICAO 72	SIMBOLO 90 - TRANSICAO 55
ESTADO : 64	ESTADO : 64
SIMBOLO 16 - TRANSICAO 73	SIMBOLO 9 - TRANSICAO 69
ESTADO : 65	ESTADO : 65
SIMBOLO 10 - TRANSICAO 53	SIMBOLO 69 - TRANSICAO 70
SIMBOLO 22 - TRANSICAO 74	
ESTADO : 66	ESTADO : 66
SIMBOLO 90 - TRANSICAO 75	SIMBOLO 5 - TRANSICAO 71
ESTADO : 67	ESTADO : 67
SIMBOLO 207 - TRANSICAO 56	SIMBOLO 16 - TRANSICAO 72
ESTADO : 68	ESTADO : 68
SIMBOLO 207 - TRANSICAO 57	SIMBOLO 13 - TRANSICAO 73
ESTADO : 69	ESTADO : 69
SIMBOLO 9 - TRANSICAO 76	SIMBOLO 81 - TRANSICAO 75
	SIMBOLO 83 - TRANSICAO 74
ESTADO : 70	ESTADO : 70
SIMBOLO 69 - TRANSICAO 77	SIMBOLO 204 - TRANSICAO 23
ESTADO : 71	ESTADO : 71
SIMBOLO 5 - TRANSICAO 78	SIMBOLO 3 - TRANSICAO 76
ESTADO : 72	ESTADO : 72
SIMBOLO 16 - TRANSICAO 79	SIMBOLO 24 - TRANSICAO 77
	SIMBOLO 25 - TRANSICAO 77
	SIMBOLO 26 - TRANSICAO 77
ESTADO : 73	ESTADO : 73
SIMBOLO 13 - TRANSICAO 80	SIMBOLO 5 - TRANSICAO 78
ESTADO : 74	ESTADO : 74
SIMBOLO 5 - TRANSICAO 65	SIMBOLO 84 - TRANSICAO 79
	SIMBOLO 85 - TRANSICAO 79
	SIMBOLO 86 - TRANSICAO 79
ESTADO : 75	ESTADO : 75
SIMBOLO 6 - TRANSICAO 81	SIMBOLO 82 - TRANSICAO 80
ESTADO : 76	ESTADO : 76
SIMBOLO 81 - TRANSICAO 83	SIMBOLO 35 - TRANSICAO 81
SIMBOLO 83 - TRANSICAO 82	SIMBOLO 36 - TRANSICAO 81
ESTADO : 77	ESTADO : 77
SIMBOLO 204 - TRANSICAO 23	SIMBOLO 5 - TRANSICAO 82
ESTADO : 78	ESTADO : 78
SIMBOLO 3 - TRANSICAO 84	SIMBOLO 17 - TRANSICAO 83
ESTADO : 79	ESTADO : 79
SIMBOLO 24 - TRANSICAO 85	SIMBOLO 82 - TRANSICAO 80
SIMBOLO 25 - TRANSICAO 85	SIMBOLO 87 - TRANSICAO 80
SIMBOLO 26 - TRANSICAO 85	
ESTADO : 80	ESTADO : 80
SIMBOLO 5 - TRANSICAO 86	SIMBOLO 7 - TRANSICAO 23
	SIMBOLO 22 - TRANSICAO 57
	SIMBOLO 205 - TRANSICAO 46
ESTADO : 81	ESTADO : 81

```
  SIMBOLO 207 - TRANSICAO  87
ESTADO :  82
  SIMBOLO  84 - TRANSICAO  88
  SIMBOLO  85 - TRANSICAO  88
  SIMBOLO  86 - TRANSICAO  88

ESTADO :  83
  SIMBOLO  82 - TRANSICAO  89
ESTADO :  84
  SIMBOLO  35 - TRANSICAO  90
  SIMBOLO  36 - TRANSICAO  90
ESTADO :  85
  SIMBOLO   5 - TRANSICAO  91
ESTADO :  86
  SIMBOLO  17 - TRANSICAO  92
ESTADO :  87
  SIMBOLO   7 - TRANSICAO  23
  SIMBOLO  10 - TRANSICAO  93
ESTADO :  88
  SIMBOLO  82 - TRANSICAO  89
  SIMBOLO  87 - TRANSICAO  89
ESTADO :  89
  SIMBOLO   7 - TRANSICAO  23
  SIMBOLO  22 - TRANSICAO  94
  SIMBOLO 205 - TRANSICAO  48
ESTADO :  90
  SIMBOLO  37 - TRANSICAO  95
ESTADO :  91
  SIMBOLO  27 - TRANSICAO  96
  SIMBOLO  28 - TRANSICAO  96
  SIMBOLO  29 - TRANSICAO  96
  SIMBOLO  30 - TRANSICAO  96
  SIMBOLO  31 - TRANSICAO  96
  SIMBOLO  32 - TRANSICAO  96
ESTADO :  92
  SIMBOLO  18 - TRANSICAO  97
ESTADO :  93
  SIMBOLO 207 - TRANSICAO  87
ESTADO :  94
  SIMBOLO  74 - TRANSICAO  98
  SIMBOLO  75 - TRANSICAO  98
  SIMBOLO  76 - TRANSICAO  98
  SIMBOLO  77 - TRANSICAO  98
  SIMBOLO  78 - TRANSICAO  98
  SIMBOLO  79 - TRANSICAO  98
  SIMBOLO  80 - TRANSICAO  98
ESTADO :  95
  SIMBOLO   5 - TRANSICAO  99
  SIMBOLO  38 - TRANSICAO  99
ESTADO :  96
  SIMBOLO  10 - TRANSICAO  53
ESTADO :  97
  SIMBOLO  19 - TRANSICAO 100
ESTADO :  98
```

```
  SIMBOLO  37 - TRANSICAO  84
ESTADO :  82
  SIMBOLO  27 - TRANSICAO  85
  SIMBOLO  28 - TRANSICAO  85
  SIMBOLO  29 - TRANSICAO  85
  SIMBOLO  30 - TRANSICAO  85
  SIMBOLO  31 - TRANSICAO  85
  SIMBOLO  32 - TRANSICAO  85
ESTADO :  83
  SIMBOLO  18 - TRANSICAO  86
ESTADO :  84
  SIMBOLO   5 - TRANSICAO  87
  SIMBOLO  38 - TRANSICAO  87
ESTADO :  85
  SIMBOLO  10 - TRANSICAO  50
ESTADO :  86
  SIMBOLO  19 - TRANSICAO  85
ESTADO :  87F
  SIMBOLO  39 - TRANSICAO  84
  SIMBOLO  40 - TRANSICAO  71
```

```
  SIMBOLO    9 - TRANSICAO 101
ESTADO :  99F
  SIMBOLO   39 - TRANSICAO 103
  SIMBOLO   40 - TRANSICAO 102
ESTADO : 100
  SIMBOLO   10 - TRANSICAO  53
ESTADO : 101
  SIMBOLO   81 - TRANSICAO 105
  SIMBOLO   83 - TRANSICAO 104
ESTADO : 102
  SIMBOLO    3 - TRANSICAO 106
ESTADO : 103
  SIMBOLO    5 - TRANSICAO  99
  SIMBOLO   38 - TRANSICAO  99
ESTADO : 104
  SIMBOLO   84 - TRANSICAO 107
  SIMBOLO   85 - TRANSICAO 107
  SIMBOLO   86 - TRANSICAO 107
ESTADO : 105
  SIMBOLO   82 - TRANSICAO  89
ESTADO : 106
  SIMBOLO   35 - TRANSICAO 108
  SIMBOLO   36 - TRANSICAO 108
ESTADO : 107
  SIMBOLO   82 - TRANSICAO  89
  SIMBOLO   87 - TRANSICAO  89
ESTADO : 108
  SIMBOLO   37 - TRANSICAO 109
ESTADO : 109
  SIMBOLO    5 - TRANSICAO 110
  SIMBOLO   38 - TRANSICAO 110
ESTADO : 110F
  SIMBOLO   39 - TRANSICAO 111
  SIMBOLO   40 - TRANSICAO 102
ESTADO : 111
  SIMBOLO    5 - TRANSICAO 110
  SIMBOLO   38 - TRANSICAO 110
ESTADOS EQUIVALENTES --> 28 29 34 50
ESTADOS EQUIVALENTES --> 45 67
ESTADOS EQUIVALENTES --> 46 68 81 93
ESTADOS EQUIVALENTES --> 55 74
ESTADOS EQUIVALENTES --> 57 87
ESTADOS EQUIVALENTES --> 58 75
ESTADOS EQUIVALENTES --> 60 94
ESTADOS EQUIVALENTES --> 69 98
ESTADOS EQUIVALENTES --> 76101
ESTADOS EQUIVALENTES --> 78102
ESTADOS EQUIVALENTES --> 82104
ESTADOS EQUIVALENTES --> 83105
ESTADOS EQUIVALENTES --> 84106
ESTADOS EQUIVALENTES --> 88107
ESTADOS EQUIVALENTES --> 90108
ESTADOS EQUIVALENTES --> 95103109111
ESTADOS EQUIVALENTES --> 96100
ESTADOS EQUIVALENTES --> 99110
```

```
AUTOMATO PRODUCAO #    3
  ESTADO :  88
    SIMBOLO  23 - TRANSICAO  89
    SIMBOLO  91 - TRANSICAO  90
  ESTADO :  89
    SIMBOLO   5 - TRANSICAO  92
    SIMBOLO  93 - TRANSICAO  91
  ESTADO :  90F
    SIMBOLO  92 - TRANSICAO  93
  ESTADO :  91
    SIMBOLO  94 - TRANSICAO  94
  ESTADO :  92
    SIMBOLO  93 - TRANSICAO  91
  ESTADO :  93
    SIMBOLO   5 - TRANSICAO  95
  ESTADO :  94
    SIMBOLO   6 - TRANSICAO  96
  ESTADO :  95F
  ESTADO :  96
    SIMBOLO   7 - TRANSICAO  95
```

```
AUTOMATO MINIMO
  ESTADO :  88
    SIMBOLO  23 - TRANSICAO  89
    SIMBOLO  91 - TRANSICAO  90
  ESTADO :  89
    SIMBOLO   5 - TRANSICAO  92
    SIMBOLO  93 - TRANSICAO  91
  ESTADO :  90F
    SIMBOLO  92 - TRANSICAO  93
  ESTADO :  91
    SIMBOLO  94 - TRANSICAO  94
  ESTADO :  92
    SIMBOLO  93 - TRANSICAO  91
  ESTADO :  93
    SIMBOLO   5 - TRANSICAO  95
  ESTADO :  94
    SIMBOLO   6 - TRANSICAO  96
  ESTADO :  95F
  ESTADO :  96
    SIMBOLO   7 - TRANSICAO  95
```

```
AUTOMATO PRODUCAO #    4
  ESTADO :  97
    SIMBOLO    5 - TRANSICAO 101
    SIMBOLO    9 - TRANSICAO  98
    SIMBOLO   38 - TRANSICAO 100
    SIMBOLO   95 - TRANSICAO 100
    SIMBOLO   96 - TRANSICAO  99
    SIMBOLO   97 - TRANSICAO  99
    SIMBOLO   98 - TRANSICAO 102
    SIMBOLO  208 - TRANSICAO 101
  ESTADO :  98
    SIMBOLO    5 - TRANSICAO 100
  ESTADO :  99F
    SIMBOLO   92 - TRANSICAO 103
    SIMBOLO   93 - TRANSICAO 104
  ESTADO : 100F
    SIMBOLO   93 - TRANSICAO 104
  ESTADO : 101F
    SIMBOLO   93 - TRANSICAO 105
  ESTADO : 102
    SIMBOLO   99 - TRANSICAO 106
  ESTADO : 103
    SIMBOLO    5 - TRANSICAO 100
  ESTADO : 104
    SIMBOLO    5 - TRANSICAO 101
    SIMBOLO   98 - TRANSICAO 102
    SIMBOLO  208 - TRANSICAO 101
  ESTADO : 105
    SIMBOLO    5 - TRANSICAO 101
    SIMBOLO   98 - TRANSICAO 107
    SIMBOLO  208 - TRANSICAO 101
  ESTADO : 106
    SIMBOLO    5 - TRANSICAO 108
  ESTADO : 107
    SIMBOLO   99 - TRANSICAO 109
  ESTADO : 108
    SIMBOLO  100 - TRANSICAO 110
  ESTADO : 109
    SIMBOLO    5 - TRANSICAO 111
  ESTADO : 110
    SIMBOLO    5 - TRANSICAO 101
    SIMBOLO  208 - TRANSICAO 101
  ESTADO : 111
    SIMBOLO  100 - TRANSICAO 112
  ESTADO : 112
    SIMBOLO    5 - TRANSICAO 101
    SIMBOLO  208 - TRANSICAO 101
  ESTADOS EQUIVALENTES --> 98103
  ESTADOS EQUIVALENTES -->100101
  ESTADOS EQUIVALENTES -->102107
  ESTADOS EQUIVALENTES -->104105
  ESTADOS EQUIVALENTES -->106109
  ESTADOS EQUIVALENTES -->108111
  ESTADOS EQUIVALENTES -->110112
```

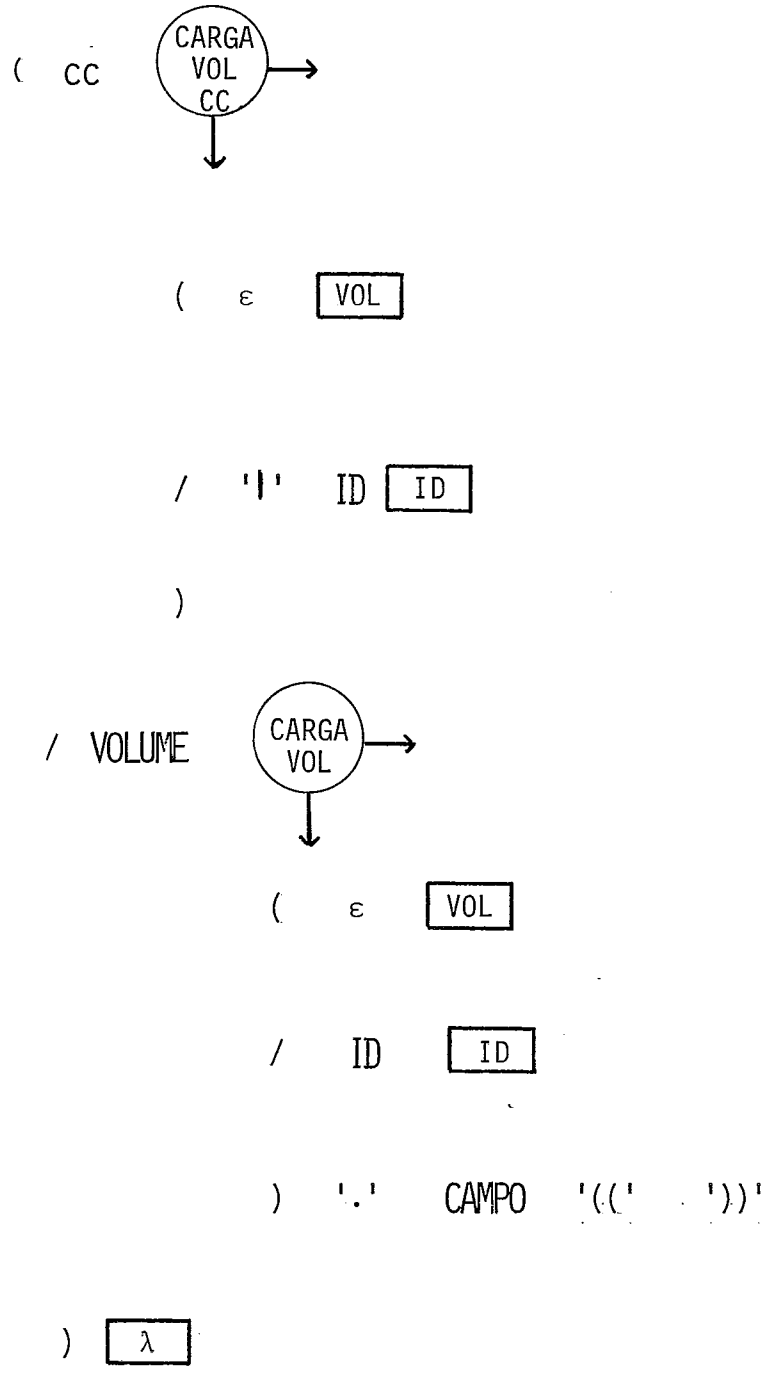
```
AUTOMATO MINIMO
  ESTADO :  97
    SIMBOLO    5 - TRANSICAO 100
    SIMBOLO    9 - TRANSICAO  98
    SIMBOLO   38 - TRANSICAO 100
    SIMBOLO   95 - TRANSICAO 100
    SIMBOLO   96 - TRANSICAO  99
    SIMBOLO   97 - TRANSICAO  99
    SIMBOLO   98 - TRANSICAO 101
    SIMBOLO  208 - TRANSICAO 100
  ESTADO :  98
    SIMBOLO    5 - TRANSICAO 100
  ESTADO :  99F
    SIMBOLO   92 - TRANSICAO  98
    SIMBOLO   93 - TRANSICAO 102
  ESTADO : 100F
    SIMBOLO   93 - TRANSICAO 102
  ESTADO : 101
    SIMBOLO   99 - TRANSICAO 103
  ESTADO : 102
    SIMBOLO    5 - TRANSICAO 100
    SIMBOLO   98 - TRANSICAO 101
    SIMBOLO  208 - TRANSICAO 100
  ESTADO : 103
    SIMBOLO    5 - TRANSICAO 104
  ESTADO : 104
    SIMBOLO  100 - TRANSICAO 105
  ESTADO : 105
    SIMBOLO    5 - TRANSICAO 100
    SIMBOLO  208 - TRANSICAO 100
```

```
AUTOMATO PRODUCAO #    5
  ESTADO : 106
    SIMBOLO 101 - TRANSICAO 108
    SIMBOLO 210 - TRANSICAO 107
  ESTADO : 107F
    SIMBOLO  40 - TRANSICAO 109
    SIMBOLO 102 - TRANSICAO 109
    SIMBOLO 103 - TRANSICAO 109
    SIMBOLO 104 - TRANSICAO 109
  ESTADO : 108
    SIMBOLO 210 - TRANSICAO 107
  ESTADO : 109
    SIMBOLO 101 - TRANSICAO 110
    SIMBOLO 210 - TRANSICAO 107
  ESTADO : 110
    SIMBOLO 210 - TRANSICAO 107
  ESTADOS EQUIVALENTES -->106109
  ESTADOS EQUIVALENTES -->108110
```

```
AUTOMATO MINIMO
  ESTADO : 106
    SIMBOLO 101 - TRANSICAO 108
    SIMBOLO 210 - TRANSICAO 107
  ESTADO : 107F
    SIMBOLO  40 - TRANSICAO 106
    SIMBOLO 102 - TRANSICAO 106
    SIMBOLO 103 - TRANSICAO 106
    SIMBOLO 104 - TRANSICAO 106
  ESTADO : 108
    SIMBOLO 210 - TRANSICAO 107
```

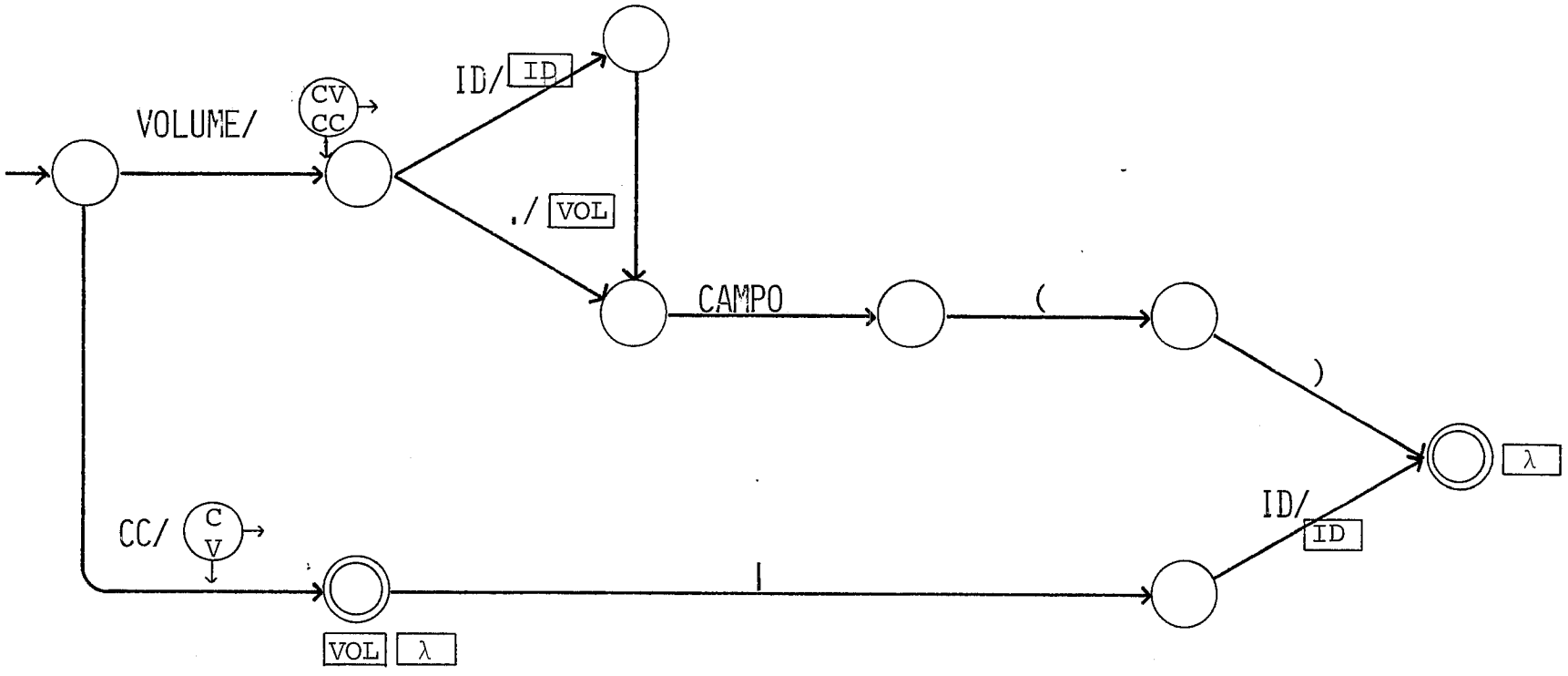
5. ÁRVORES DE CÓDIGO

3. end - campo : : =



6. GRAMÁTICA EM FORMA DE AFD COM AS ÁRVORES DE CÓDIGO

3. END - CAMPO



X. BIBLIOGRAFIA

1. SANTOS, A.C. et al., ESPECIFICAÇÃO DA LINGUAGEM LOBAN,1980 Versão 2, Relatório Técnico, COPPE/UFRJ, 1981.

2. SOUSA, A.C. & MIYASATO, B.Z. & SIMONE, E.G. & SOUZA, J.M. DANTAS, J.S. & CRIVOROT, S.H. & D'ALBUQUERQUE, V.L. O Banco de dados Microloban. ANAIS DO IX SEMINÁRIO INTEGRADO DE SOFTWARE E HARDWARE , OURO PRETO , 1982

3. SIMONE, E.G. & PEREIRA, L.C. - ALGORITMOS PARA GRAMÁTICAS RRP-SLR(1). ANAIS DO VII SEMINÁRIO INTEGRADO DE SOFTWARE E HARDWARE , Campinas , 1980

4. GRIFFITHS, M - LL(1) Grammars and Analysers in: BAUER, F.L. & EICKEL, J. - COMPILER CONSTRUCTION AN ADVANCED COURSE. 2.Ed. New York, N.Y., Springer-Verlag, 1974, Capítulo 2.B., P. 57-83

5. TELLES, A.A.S. & SIMONE, E.G. - Gerador de Analisadores Sintáticos RRP-LL(1). ANAIS DO VIII SEMINÁRIO INTEGRADO DE SOFTWARE E HARDWARE , Florianópolis , 1981.

6. AHO, A.V. & ULLMAN, J.D. - THE THEORY OF PARSING, TRANSLATION AND COMPILING , Volume 1 : PARSING. Englewood Cliffs, N.J., Prentice-Hall, 1972

7. ARGOLLO JR, M.T. & SIMONE, E.G. - CODIFICADOR RRP: MANUAL DE LÓGICA. Relatório técnico ES-11-81 , Programa de Engenharia de sistemas e computação, COPPE, UFRJ, 34 P.

8. AHO, A.V. & ULLMAN, J.D. - PRINCIPLES OF COMPILER DESIGN 2.Ed. Reading , Mass., Addison-Wesley, 1977

9. LEWIS II, P.M. & STEARNS, R.E. - Syntax-Directed Transduction. JOURNAL OF THE ASSOCIATION FOR COMPUTING MACHINERY , VOL. 15, Nº 3 , JULHO 1968, P.465-488.

10. D'ALBUQUERQUE, V.L. - INTERPRETADOR MICROLOBAN. Tese de M.SC., COPPE UFRJ , em preparação.

11. KNUTH, D.E. - THE ART OF COMPUTER PROGRAMMING. Volume 1 FUNDAMENTAL ALGORITHMS - World Student series edition 2.ed. Reading, Mass. , Addison-Wesley , 1976

12. ARGOLLO JR, M.T. & SIMONE, E.G. - CODIFICADOR RRP: MANUAL UTILIZAÇÃO. Relatório técnico ES-10-81 , Programa de Engenharia de Sistemas e Computação, COPPE , UFRJ, 20 P.

13. GRAHAM, S.L. & RHODES, S.P. - Practical Syntactic Error Recovery, COMMUNICATIONS OF THE ACM , Novembro 1975, Volume 18, N. 11 , P. 639-650