

LINGUAGEM DE DESCRIÇÃO DE ROTINAS:

PARTE DE CONTROLE

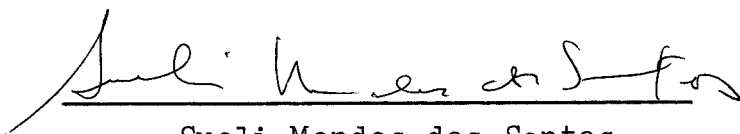
Carlos Flores Cunha

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.)

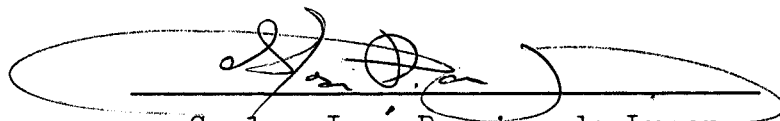
Aprovada por:



Paulo Augusto Silva Veloso  
(Presidente)



Sueli Mendes dos Santos



Carlos José Pereira de Lucena

RIO DE JANEIRO, RJ - BRASIL

FEVEREIRO DE 1979

CUNHA, CARLOS FLORES

Linguagem de Descrição de Rotinas: Parte de Controle [Rio de Janeiro] 1979.

ix, 62 p. 29,7cm(COPPE-UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1979)

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

1. Linguagens de Programação I. COPPE/UFRJ II. Título(série)

Copyright ©1979 Carlos Flores Cunha

É permitida cópia de todo ou parte deste trabalho sem ônus desde que:

- a - não exista envolvimento comercial direto ou indireto,
- b - sejam citados os dados bibliográficos completos deste trabalho e
- c - seja citado este aviso de copyright.

Qualquer outra forma de utilização, apenas com permissão por escrito do autor.

Carlos Flores Cunha

Rua Ronald de Carvalho 292 ap.902

22021 Rio de Janeiro

Tel. (021) 257-2336

DEDICATÓRIA:

Aos que sentem,

têm esperanças

e lutam.

AGRADECIMENTOS :

Carlos Alberto da S. Franco

Guilherme Chagas Rodrigues

José Carlos Garcia da Costa

Paulo Augusto Silva Veloso

Walter Dominguez

e

CELIA ABICALIL BELMIRO

SINOPSE:

Iniciamos este trabalho abordando tabelas de decisões nas possibilidades e vantagens de seu uso, sua importância como recurso técnico e linguagem.

Deste ponto, partimos para a elaboração de sintaxes, objetivando um melhor aproveitamento de tabelas de decisões em programação e descrição de rotinas.

Uma linguagem de programação pode se estruturar em três áreas : especificações, controle, operações. Centramos-nos na área de controle, destacando dois estágios de sua estrutura sintática : o primeiro, relação entre os componentes, em que se definem sete sintaxes - básica, reduzida, com preâmbulo, ações paralelas, transições paralelas, ações não-determinísticas, transições não-determinísticas. No segundo estágio, são detalhados os componentes do primeiro estágio, incluindo as definições sintáticas de tabelas de decisões com entrada limitada e entrada mista.

A linguagem básica é extensamente trabalhada, servindo de suporte às outras. Fazem parte de seu estudo o processamento de uma tabela, suas propriedades e alguns aspectos que podem ser vistos em uma estrutura de controle. Das demais linguagens serão descritas somente os elementos que dão o traço específico a cada uma.

Preocupamo-nos em levantar, ainda, um material bibliográfico atinente a ta be la de decisões (Anexo B).

ABSTRACT:

We browse the possibilities and advantages of the usage of decision tables, their importance as a technical tool and as a language.

The kernel of this work is the suggestion of languages for programming and description of routines using tables.

We view a programming language as composed of specifications, control and operations and we only deal with the control area, emphasizing its syntactical structure on two levels. The one, relationships among components, where seven syntaxes are outlined - basic, reduced, with preamble, parallel actions, parallel transitions, nondeterministic actions, nondeterministic transitions. The other, component description, describes limited-entry and mixed-entry decision tables syntactics.

The basic language is widely dealt with, standing as a support for all the others. Table processing, its characteristic, and some aspects which can be examined within a control structure are part of its study. For the other languages, only the elements which would provide their specific features are described.

"Anexo B" contains a bibliography on decision tables.

Índice:

- 1 Introdução
- 2 Divisões
- 3 Controle
- 4 Sintaxes
- 5 Linguagem número 1 - Linguagem básica
  - 5.1 Processamento de uma tabela
  - 5.2 Determinação da regra a partir da avaliação das condições
  - 5.3 Determinação das ações e transição
  - 5.4 Exemplo de processamento de uma tabela
  - 5.5 Tabela sem condições e regras
  - 5.6 Propriedade das tabelas :
    - 5.6.1 Uma tabela tem que ser completa
    - 5.6.2 Uma tabela não pode ter regras incompatíveis
    - 5.6.3 As regras podem ser redundantes
    - 5.6.4 A ordem das regras é irrelevante
    - 5.6.5 A ordem das condições é irrelevante
- 6 Estrutura sintática da linguagem número 1
  - 6.1 Relação entre os componentes (primeiro nível) :
    - 6.1.1 Sintaxe-padrão (diagrama de transições)
    - 6.1.2 Sintaxe completa (diagrama de transições)
    - 6.1.3 Sintaxe-padrão (tabela de transições)
    - 6.1.4 Sintaxe completa (tabela de transições)
  - 6.2 Definição de componentes (segundo nível) :
    - 6.2.1 comentários
    - 6.2.2 identificação
    - 6.2.3 parâmetros
    - 6.2.4 condições
    - 6.2.5 condição
    - 6.2.6 relação

6.2.7 regra

6.2.8 transição

6.2.9 transição em tabela com condições

6.2.10 transição em tabela sem condições

6.2.11 ações

6.2.12 atribuição

6.3 Exemplo (pesquisa em tabela)

6.4 Exemplo (greve dos metalúrgicos)

6.5 Exemplo (baseado em [KING 69])

7 Linguagem para tabela de decisões com entrada mista

7.1 Entrada mista

7.2 Estrutura sintática para tabelas de decisão com entrada mista

7.2.1 Relação entre os componentes :

7.2.2 Definição dos componentes :

7.2.2.1 condição

7.2.2.2 relação

7.2.2.3 regra

7.3 Exemplo (baseado em [FISHER 66])

8 Linguagem número 2 - linguagem reduzida

8.1 Quantidade de símbolos

8.2 Relação entre os componentes :

8.2.1 Sintaxe-padrão

8.2.2 Sintaxe completa

9 Linguagem número 3 - com preâmbulo

9.1 Tabela com preâmbulo

9.2 Relação entre os componentes :

9.2.1 Sintaxe-padrão (Figura x)

9.2.2 Sintaxe completa (Figura y)

9.3 Exemplo

- 10 Linguagem número 4 - com ações paralelas
  - 10.1 Ações paralelas
  - 10.2 Relações entre os componentes :
    - 10.2.1 Sintaxe-padrão (Figura x)
    - 10.2.2 Sintaxe completa (Figura y )
  
- 11 Linguagem número 5 - com transições paralelas
  - 11.1 Transições paralelas
    - 11.1.1 Bifurcação
    - 11.1.2 Junção
  - 11.2 Estrutura sintática da linguagem número 5
    - 11.2.1 Relação entre componentes :
      - 11.2.2.1 Sintaxe-padrão (Figura x)
      - 11.2.1.2 Sintaxe completa (Figura y)
    - 11.2.2 Definição dos componentes :
      - 11.2.2.1 identificação
      - 11.2.2.2 transição
      - 11.2.2.3 transição paralela
  
- 12 Linguagem número 6 - com ações não-determinísticas
  - 12.1 Não-determinismo
  - 12.2 Ações não-determinísticas
  - 12.3 Relação entre os componentes :
    - 12.3.1 Sintaxe-padrão (Figura x)
    - 12.3.2 Sintaxe completa (Figura y)
  
- 13 Linguagem número 7 - com transições não-determinísticas
  - 13.1 Transições não-determinísticas
  - 13.2 Estrutura sintática da linguagem número 2
    - 13.2.1 Relação entre os componentes :
      - 13.2.1.1 Sintaxe-padrão (Figura x)
      - 13.2.1.2 Sintaxe completa (Figura y)
    - 13.2.2 Definição dos componentes :
      - 13.2.2.1 transição não-determinísticas
  
- 14 Conclusões

## 1 - Introdução

Há mais de duas décadas, o uso de tabelas de decisões tem sido bastante difundido no desenvolvimento e documentação de programas, existindo uma vasta bibliografia sobre o assunto (ver levantamento bibliográfico Anexo B). Quanto mais complexo for o programa, quanto mais crescer a quantidade de decisões envolvidas em uma lógica complexa, mais benefícios serão encontrados no emprego de tabelas de decisões [FISHER 66 , CANNING 72, METZNER 77].

Em [FISHER 66] é citada a dificuldade que a IBM teve, em 1958, para definir um problema extremamente complexo de manutenção de arquivo. Foram gastos, sem sucesso, quase sete homens-ano tentando definir o problema, usando fluxogramas e narrativas. Com o emprego de tabelas de decisões, conseguiram definir o problema com doze homens-semana. Fisher acredita que, apenas com o uso de fluxogramas e narrativas, mesmo não havendo limitação de tempo, não dispunham de meios suficientes para chegar a definir o problema. Somente com o emprego de tabelas de decisões, passaram a possuir recursos capazes de fazer esta definição.

Em [DEAN 71] aparece o resultado de um estudo feito na Inglaterra envolvendo cinco instalações de processamento de dados. Nesta experiência, mediu-se o tempo de produção de aproximadamente noventa programas, nos quais se utilizou a técnica de tabelas de decisões. Comparou-se este tempo com o que seria gasto com as formas de programação normalmente utilizadas nestas instalações. A relação entre estes tempos mostrou que o uso de tabelas de decisões provocou um aumento da produtividade na faixa de 180% a 350%.

No início da década de 60, pessoas envolvidas em grandes projetos de programação, [CENSUS 64] por exemplo, acreditaram que tabelas de decisões tinham surgido como uma grande solução para formalizar as informações entre administradores e analistas, entre analistas e programadores e como uma técnica muito potente para desenvolver e documentar programas.

Contrariamente ao previsto, a sua utilização não é tão ampla por parte dos que estão ligados a processamento de dados.

Diversos fatores parecem ter influído na maior ou menor aceitação do emprego de tabelas de decisões. [WEGNER 78] se refere à reação ao emprego efetivo de novas linguagens. Quando Fortran e Cobol apareceram, encontraram uma grande área vazia e sedenta de um melhor instrumental de programação. Como não tinham concorrência por parte de outras linguagens, sua difusão foi muito ampla e rápida, ocupando todos os espaços possíveis. Mesmo não satisfazendo plenamente, eram utilizadas por serem os recursos disponíveis.

Até hoje, esta tensão permanece. Apesar do assédio constante que sofrem cada vez mais pelo desenvolvimento contínuo de novas e potentes linguagens, a maioria dos programas estão e continuam sendo escritos em Fortran e Cobol.

Outra situação que vem dificultando a popularização do uso de tabelas das decisões é o seu emprego, até por parte de seus defensores maiores, apenas como um meio auxiliar de elaboração de programas: primeiro devem ser construídas as tabelas de decisões e, a partir daí, codificar os programas. Portanto, estão dando às tabelas de decisões o mesmo papel que tem os diagramas de blocos. Em [POOCH 74] por exemplo, existe uma boa comparação entre tabelas de decisões e diagramas de blocos.

Foram desenvolvidas diversas linguagens que usam tabelas de decisões e, em todas elas, a linguagem inicial é traduzida para alguma linguagem de programação já existente, geralmente Cobol ou Fortran, havendo também traduções para Algol, PL/1, Basic e Lisp (ver Anexo T). Nenhuma destas linguagens existe como linguagem independente, persistindo o conceito de tabelas de decisões como uma forma gráfica auxiliar para a construção de programas, função de diagrama de blocos ainda. A diferença se faz, agora, na conversão automática para uma linguagem de programação. Um inconveniente de utilizar linguagens com estas características é a necessidade de dominar duas linguagens: a linguagem

que usa tabelas e aquela em que a primeira é traduzida. Outro inconveniente é a necessidade de dupla tradução, havendo em cada uma delas uma análise e geração de novo código. Conseqüentemente, os erros que porventura ocorrerem na criação do programa original, só poderão ser detectados em duas fases, cada uma analisando um conjunto diferente de erros e sendo elas obrigatoriamente consecutivas.

O objetivo a que nos propomos neste trabalho é definir concretamente uma linguagem para o uso de tabelas, sem necessidade de conversões para outra linguagem de processamento de dados.

Para tal, foi desenvolvido um conjunto de sintaxes com uma preocupação eminentemente prática, visando facilitar e simplificar as tarefas desenvolvidas pelos profissionais que atuam em análise e programação.

As sintaxes, aqui propostas, também podem ser empregadas em qualquer tipo de rotina, estando incluído como rotina qualquer conjunto de ações que devam ser executadas segundo alguma organização.

Esperamos que as sintaxes apresentadas sejam úteis não apenas para as pessoas diretamente ligadas a processamento de dados, mas a todas que, de uma maneira ou outra, tenham algum benefício com a utilização de tabelas de decisões.

## 2 - Divisões

Existem várias formas de descrever rotinas.

Conseguimos identificar, nelas, três áreas :

- especificações - descrevem as características dos recursos disponíveis,

- operações - dividem-se em dois tipos :

1º) condições - indicações do que deve ser avaliado, só admitindo como respostas sim ou não,

2º) ações - indicam o que deve ser executado,

- controle - é a parte que indica qual caminho que a rotina deve seguir.

As linguagens para descrição de rotinas devem ser construídas em partes separadas, obedecendo esta divisão. Teremos, assim, muito maior flexibilidade no uso destas linguagens. Quando alguém quiser descrever uma rotina escolherá, de cada uma das três partes, qual é a que é mais adequada para seu caso. Atualmente, já se faz algo parecido tendendo, todavia, para uma complexidade desnecessária.

Para ter acesso a um banco de dados, utilizando uma linguagem de programação já existente, uma solução frequentemente empregada é a feitura de um programa híbrido, com instruções em duas linguagens, sendo uma específica para o banco de dados. O programa assim construído é passado por dois compiladores (ou por um pré-processador e um compilador, se preferirem). Assim, o primeiro separa e processa apenas suas instruções, gerando, a partir delas, instruções, sub-rotinas e "macros" dentro da sintaxe da segunda linguagem, escondendo suas próprias instruções na forma de comentário. Depois disto, este programa é passado para o segundo compilador.

Este processo se origina do fato de que a linguagem base (a segunda) foi construída como um bloco indivisível, isto é, já estão definidas suas

partes de especificações de dados, operações e de controle, como um todo. Como sua parte de especificação não é adequada para lidar com banco de dados, mascaram-se partes da linguagem e acrescentam-se outras. A partir daí, faz-se a ligação com outra linguagem que tem uma parte forte em especificações de dados, mas é fraca em operações e controle. Desta forma, uma série de artifícios são utilizados para "enganar" o compilador da linguagem base.

Supondo cada parte separada, não haveria necessidade de todo este esforço e cada um teria sintaxes bem mais simples e, conseqüentemente, compiladores menos complexos. Inegavelmente, os benefícios que se des<sub>u</sub> cortinariam seriam imensos. Optando pelo seu uso, cada vez que alguém precisar alterar alguma linguagem existente, bastará trabalhar as partes envolvidas, não precisando lidar com linguagens completas e fechadas. Com a construção de apenas um componente alternativo os usuários da linguagem passarão a dispor de um instrumental flexível e adaptável às situações-problema.

A seguir, deter-nos-emos mais detalhadamente no setor de controle, já que é nele que se concretizará toda uma sintaxe decorrente do uso de tabelas de decisões. Esta relação direta controle x tabelas de decisões cria uma estrutura completamente independente, que pode ser aplicada em qualquer área onde seja vantajoso o uso de tabelas de decisões.

### 3 - Controle

Descrever uma rotina é, basicamente, construir um gráfico que indique em que sequência e segundo quais condições determinadas ações devem ser executadas. Em programação, forma de rotina que já foi mais estudada, alguns defendem a idéia de que é desnecessário, ou até mesmo prejudicial, fazer explicitamente algum desenho; que o programa deve ser, logo de início, escrito em uma forma compilável ou algo parecido, seguindo ou não determinadas construções básicas. De qualquer forma, sempre existe uma construção gráfica, mesmo implícita, que descreve a sequência a seguir.

Um fato que afeta bastante o ato de programar está na própria construção de um programa - definição de uma sequência não-linear de ações - já que as linguagens disponíveis são lineares. A solução encontrada é seccionar trechos lineares do programa e colocá-los uns após os outros. Para não haver confusão no que deve ser executado, é criado uma série de parênteses, separadores e desvios (begin...end, if...else..., while ...do..., for... until..., goto... etc.).

Quando construímos um diagrama de blocos não há necessidade de separadores, por ser ele uma forma não linear de representar programas. Quanto mais a parte de controle de uma linguagem se aproximar de um gráfico, mais direta e imediata será sua compreensão e menor será a necessidade de parênteses e separadores.

Para introduzir a área de controle sugerida, vejamos a Figura 1, que é o exemplo 5 de [KNUTH 74]. É um algoritmo de inserção em árvore binária, sendo  $A[i]$  um nó,  $L[i]$  um apontador para o nó da sub-árvore esquerda e  $R[i]$  apontador para a sub-árvore direita. É suposto que será sempre feita uma inserção, isto é, o valor a ser inserido não está presente na árvore. Inicialmente,  $j$  indica a primeira posição vazia em  $A$  e  $i$  indica o nó da árvore.

Uma possível forma gráfica bidimensional é o diagrama de blocos da Figura 2. Precisamos desenvolver linguagens que sejam capazes de descrever este comportamento não-linear tanto para ser utilizado em programação como para descrever rotinas em geral. É exatamente este o ponto central do nosso trabalho, que buscamos atingir com as diversas sintaxes propos

tas para a parte de controle.

A Figura 3 é uma representação da rotina anterior, usando uma das sintaxes propostas.

Na Figura 4, temos um detalhamento da Figura 3, estando separadas as áreas de controle e de operações. A parte referente a controle não possui nenhuma palavra-chave em inglês (if, then, goto, else, begin, end etc.), ou em qualquer outro idioma. Podemos identificar, ainda, três componentes de controle, nesta tabela :

- identificação - que dá a forma de se fazer referência a este ponto da rotina;
- regras - definem possíveis conjuntos de respostas, baseados na avaliação das condições. Foi adotada a seguinte convenção: 1 (um) = sim; 0 (zero) = não; - (traço) = irrelevante;
- transições - é a parte que indica qual a próxima tabela que deve ser executada.

Na área de operações encontramos dois componentes :

- condições e
- ações.

Retomando à Figura 3, entendemos, então, que ela deve ser compreendida da seguinte forma : na parte da rotina chamada compare existem as condições  $A [i] < x$ ,  $L [i] = 0$  e  $R [i] = 0$ . Se a avaliação destas condições apresentar, respectivamente : sim, sim e qualquer valor (a terceira condição avaliada como sim ou não), será executada a ação  $L [i] := j$  e depois o controle passará para o ponto com nome de insert. Se a avaliação for sim, não, qualquer valor será executada a ação  $i := L [i]$  e irá para compare. E, assim, dependendo das condições, a regra indicará qual o conjunto de ações que deve ser executado e qual transição determina a tabela seguinte. No ponto chamado insert não existe nenhuma condição, não existindo, portanto, nenhuma regra. Serão executadas as ações correspondentes e seguirá para fim que, neste exemplo, foi escolhido para indicar o final da rotina.

```

compare :

if  A [i] < x
then if  L [i] ≠ 0
    then  i := L[i]; go to  compare;
    else  L[i] := j; go to  insert fi;
else if  R[i] ≠ 0
    then  i := R[i]; go to  compare;
    else  R [i] := j; go to  insert fi;

fi;
insert : A[j] := x;
L [j] := 0; R[j] = 0; j := j+1;

```

Obs : O símbolo fi é usado como limitador do if.

Figura 1

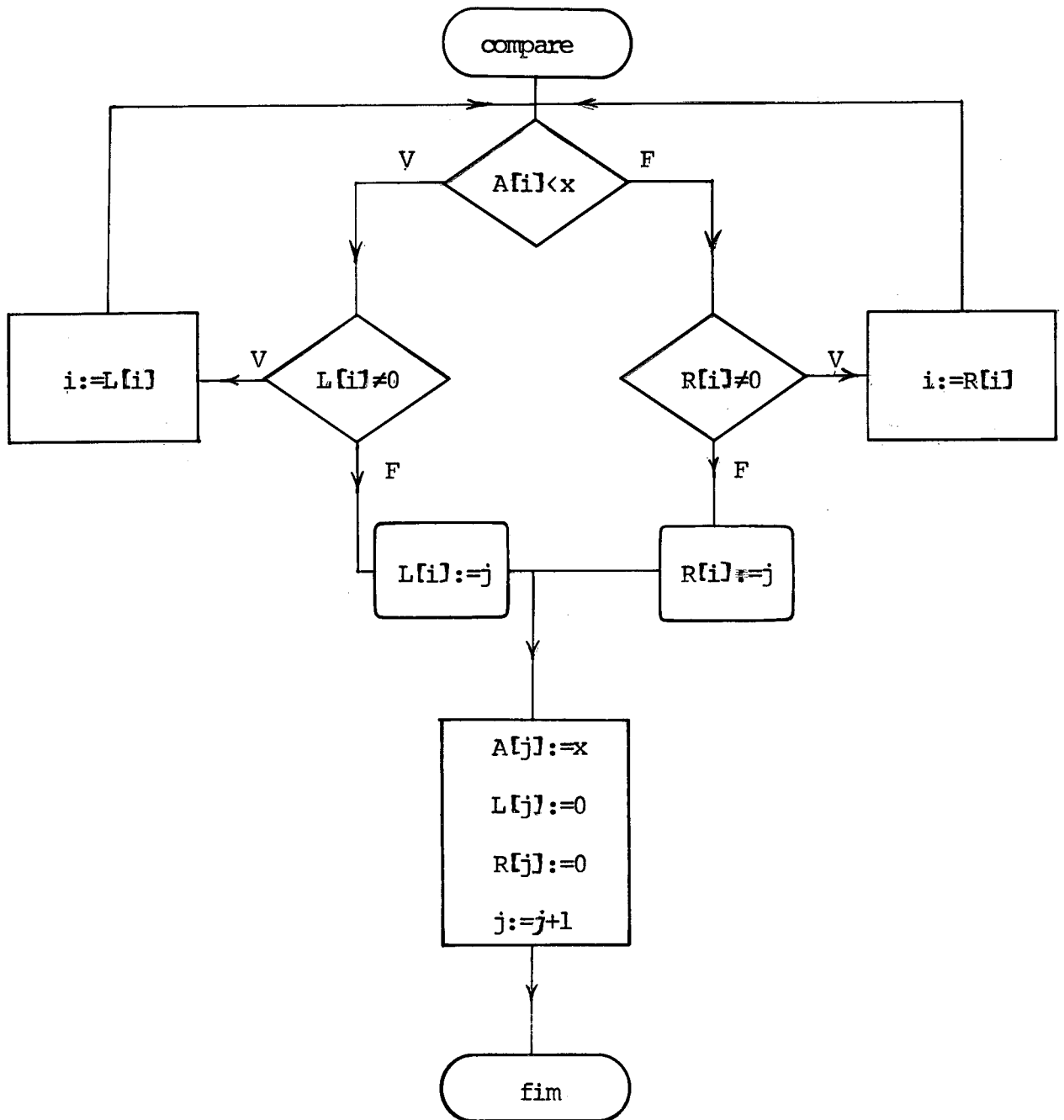


Figura 2.

```

# compare : A[i] < x . L[i] = 0 . R[i] = 0
          : 1      1      -      * insert * L[i] := j
          : 1      0      -      * compare * i := L[i]
          : 0      -      1      * insert * R[i] := j
          : 0      -      0      * compare * i := R[i] &

# insert          * fim *      A[j] := x;
                  L[j] := 0; R[j] := 0; j := j+1 &
    
```

Figura 3

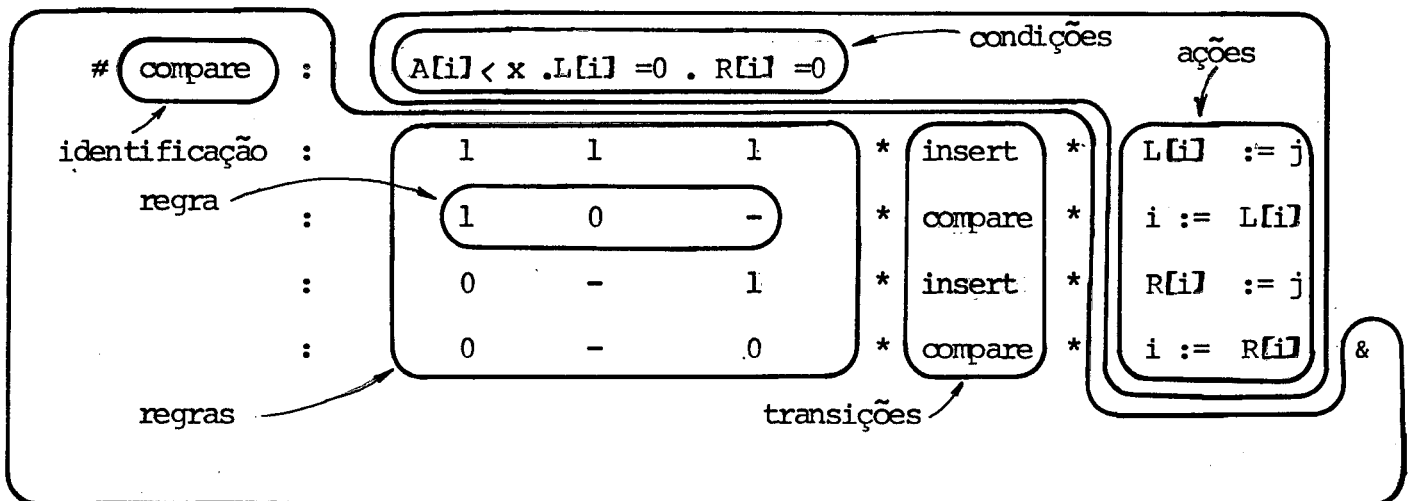


Figura 4

4 - Sintaxes

Chegamos, aqui, ao núcleo deste trabalho: definir linguagens para área de controle usando tabelas de decisões. Para tanto, apresentaremos diversas sintaxes com diferentes capacidades de descrever rotinas, quais sejam: lidar com tabelas de entrada limitada ou entrada mista; ter ou não um número reduzido de símbolos; aceitar ou não ações paralelas, transições paralelas, ações não-determinísticas e transições não-determinísticas; poder ainda ter preâmbulo e nele ser aceito paralelismo ou não-determinismo. Com todas as combinações destas características, obteríamos várias centenas de linguagens. Aqui especificamos um reduzido número delas, sete sintaxes, das quais sugerimos a implantação de apenas cinco.

A primeira sintaxe já é constituída de todas as características e vantagens necessárias para utilizar tabelas de decisões. Contudo, aqui a situamos apenas como linguagem básica inicial na compreensão e detalhamento das outras linguagens. A segunda sintaxe funciona como meio para uma breve discussão sobre o número de símbolos utilizados. Na definição das linguagens, são fornecidos diversos componentes sintáticos que, combinados, criam linguagens com características adequadas a seu universo de atuação. Entendemos que a sequência de sintaxes de 3 a 7, por nós apresentada, atende ao nosso objetivo de obter gradualmente linguagens mais potentes.

Para cada combinação de fatores que caracterizam a potência de uma linguagem, serão apresentadas, na realidade, duas linguagens: uma mais formal (que chamamos de linguagem-padrão) e outra para ser realmente implantada e utilizada (linguagem completa). A linguagem-padrão é descrita por uma sintaxe-padrão que define todas as sequências corretas de símbolos. A linguagem-completa, descrita por uma sintaxe completa, é formada por todas as possíveis sequências de símbolos, corretas ou não. Esta sintaxe descreve os casos que serão aceitos fora da sintaxe-padrão e qual a forma de serem aceitos. Indica, também, como será analisado o resto da rotina quando aparecer uma sequência não-aceita (erro). Quando alguém escrever uma rotina, utilizará sempre a sintaxe completa mas, quando fizer alguma consulta, terá acesso à rotina escrita segundo a sintaxe-padrão.

As linguagens apresentadas são todas de formato livre, vale dizer, não é necessário alinhar as tabelas, só interessando a sequência correta. Porém, quando uma tabela for consultada, aparecerá sempre alinhada.

Na estruturação das sete sintaxes concorrem seis componentes: 1 - comentários, 2 - identificação, 3 - condições, 4 - regra, 5 - transição, 6 - ações, que são separados por dois a nove símbolos diferentes, de acordo com as características da linguagem definida. O mesmo símbolo precederá sempre cada componente em cada sintaxe. A Figura 5 mostra esta correspondência nas sintaxes de número 3 a 7.

A ordenação destes componentes e símbolos é realizada em forma de máquinas sequenciais. Estas máquinas estão divididas em dois níveis. No primeiro nível, os estados são os seis componentes da linguagem, encarados como sub-máquinas e o alfabeto de entrada é o conjunto de símbolos que separam estes componentes. A Figura 6 é o resumo das sete linguagens com suas características, número de símbolos utilizados e quantidade de estados.

No segundo nível, detalhamos identificação, regra e transição que pertencem à área de controle. Fazemos também referências a condições e ações, mereas sugestões a quem estudar a área de operações.

As máquinas sequenciais serão descritas, aqui, através de diagramas de transições e tabelas de transições. Utilizaremos tabelas de transições para descrever o primeiro nível de todas as sintaxes e diagramas de transições para o segundo nível. À guisa de comparação, apresentaremos diagramas e tabelas de transições, no primeiro nível da primeira sintaxe.

Descreveremos inicialmente, a sintaxe básica (sintaxe número 1). Quanto às outras sintaxes, trabalharemos somente suas diferenças em relação à primeira e só a elas faremos referências.

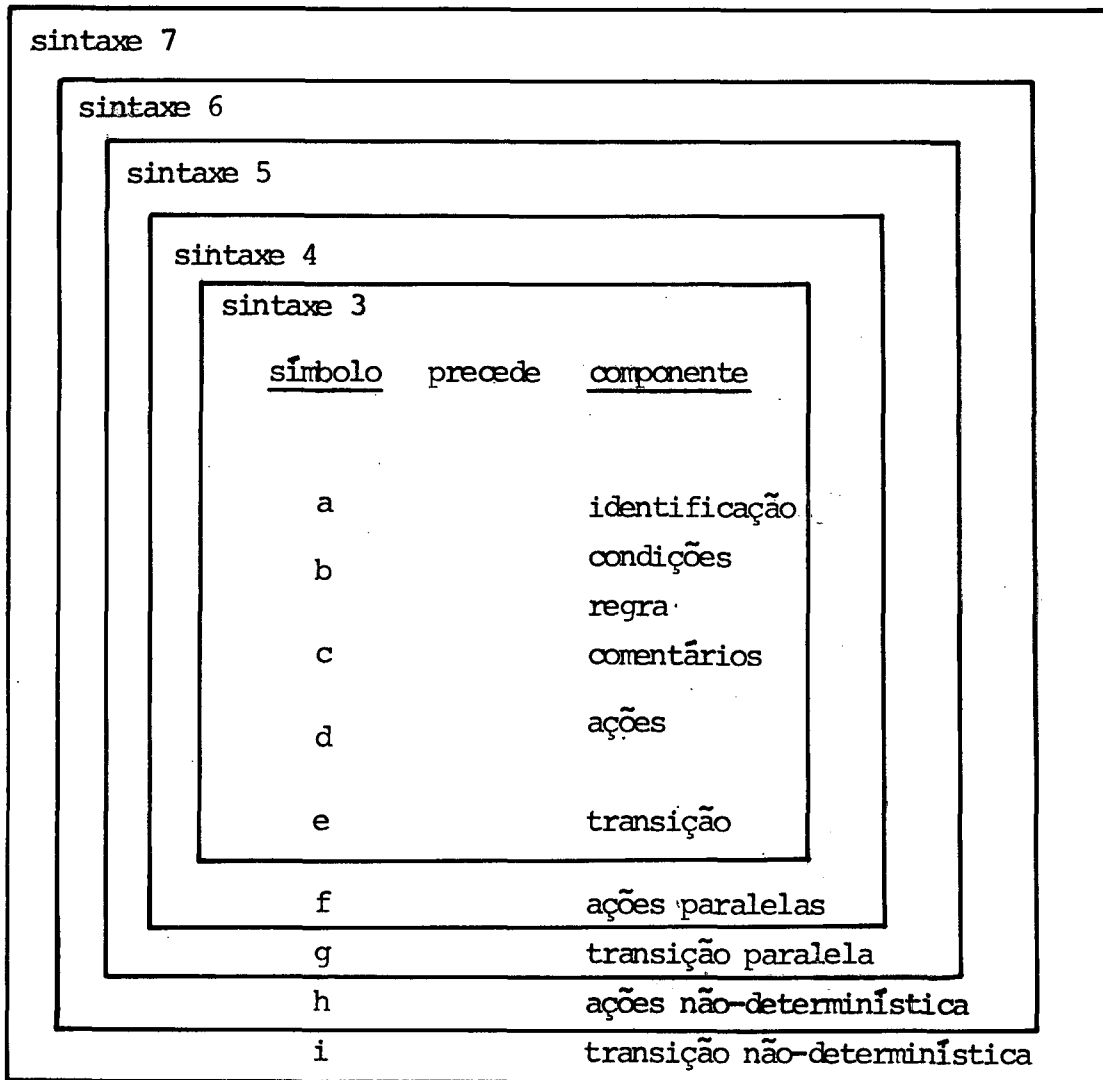


Figura 5

linguagem	número de símbolos	quantidade de estados	características das linguagens							
			preâmbulo		paralelismo		não-determinismo			
			preâmbulo	ação	transição	preâmbulo	ação	transição		
1	4	8								
2	2	8								
3	5	9	X							
4	6	12	X	X						
5	7	18	X	X	X					
6	8	23	X	X	X	X		X	X	
7	9	31	X	X	X	X	X	X	X	X

Figura 6

## 5 - Linguagem número 1 - Linguagem básica

### 5.1 - Processamento de uma tabela

Quando uma rotina alcança uma tabela, são avaliadas as condições e criada uma lista com uma quantidade de elementos igual ao número de condições. Esta lista é constituída por uma sequência de três símbolos:

- 1 (um), quando a resposta da condição for sim,
- 0 (zero), quando a resposta da condição correspondente for não,
- (traço), quando, por alguma razão, não for possível avaliar a condição correspondente (quando um elemento de um vetor estiver fora de seus limites, divisão por zero etc.).

Ao dispormos desta lista de avaliações, partiremos para a determinação da regra que a prevê. Será, então, executada a parte de ações que corresponde a esta regra e, após a conclusão da execução das ações, a rotina segue para a tabela que está indicada em transição.

### 5.2 - Determinação da regra a partir da avaliação das condições

Uma regra tem o mesmo número de elementos que a lista de condições e os mesmos símbolos que a avaliação desta lista, porém com um significado um pouco diferente :

- 1 (um), condição avaliada como sim,
- 0 (zero), quando a resposta da condição for não,
- (traço), condição que não precisa ser avaliada e, portanto, qualquer resposta é aceita.

Uma lista de avaliações de condições determina uma regra, na medida em que para cada símbolo 1 (um), nesta lista, a regra possua o símbolo 1 (um) ou - (traço) na posição correspondente; para cada símbolo 0 (zero) a regra possua 0 (zero) ou - (traço); para cada símbolo - (traço) a regra tenha o símbolo - (traço).

### 5.3 - Determinação das ações e transição

Para cada regra existe apenas um conjunto de ações e uma transição, que são os que aparecem imediatamente após a regra. Porém, várias re-

gras podem determinar um mesmo conjunto de ações e uma mesma transição.

#### 5.4 - Exemplo de processamento de uma tabela :

$$\begin{array}{l} \# \quad t_2 : c_1 \cdot c_2 \\ \quad \quad : 1 \quad - \\ \quad \quad : - \quad 1 \quad * t_1 * \quad a_1 \\ \quad \quad : 0 \quad 0 \quad * t_2 * \quad a_2 \quad \& \end{array}$$

onde  $c_1$  e  $c_2$  são condições,  $a_1$  e  $a_2$  são seqüências de ações e  $t_1$  e  $t_2$  são identificadores de tabelas. Se estiver sendo executada uma rotina e ela chegar à tabela  $t_2$ , a primeira coisa a ser feita é a avaliação das condições  $c_1$  e  $c_2$ . O próximo passo dependerá da avaliação destas duas condições. Vejamos algumas possíveis avaliações :

- condições  $c_1$  e  $c_2$  avaliadas como 00 (a resposta a  $c_1$  foi não e a  $c_2$  também foi não) : esta avaliação determina a terceira regra (00), que aponta para as ações  $a_2$  e transição  $t_2$ . O próximo passo, então, é a execução das ações  $a_2$  e, depois, a rotina segue para a tabela indicada. No caso, a transição é  $t_2$ , isto é, no exemplo apresentado, volta a processar a mesma tabela;
- condições  $c_1$  e  $c_2$  avaliadas como 01 : será determinada a segunda regra (-1). Serão executadas as ações  $a_1$  e depois a rotina seguirá para a tabela  $t_1$ ;
- condições  $c_1$  e  $c_2$  avaliadas como 10 : será determinada a primeira regra (1-). Serão executadas as ações  $a_1$  e depois a rotina seguirá para a tabela  $t_1$  (notar que a primeira e a segunda regra estão associadas às mesmas ações e à mesma transição);
- condições  $c_1$  e  $c_2$  avaliadas como 1- (condição  $c_1$  avaliada como sim e não foi possível avaliar a condição  $c_2$ ) : será determinada a primeira regra (1-);
- condições  $c_1$  e  $c_2$  avaliadas como -0 (não foi possível avaliar a condição  $c_1$  e a  $c_2$  foi avaliada como não) : este caso é indicado como erro, pois não é possível associar esta avaliação a nenhuma regra desta tabela.

### 5.5 - Tabela sem condições e regras

Se, em uma tabela, não houver os componentes condições e regras, teremos somente um conjunto de ações e uma transição. Neste caso, não há o que avaliar ou decidir e, naturalmente, as ações serão executadas e a rotina segue para a transição indicada.

### 5.6 - Propriedades das tabelas :

#### 5.6.1 - Uma tabela tem que ser completa

Toda possível lista de avaliações das condições que possua apenas os símbolos 0 e 1 tem que determinar, pelo menos, uma regra. Quando for impossível ocorrer alguma combinação de avaliação, é preciso que fique explícito. Exemplo :

```
# t : a < 2 . a > 4
      : 1      1      *      *
      : 1      0      * t1 * a2
      : 0      1      * t2 * a4
      : 0      0      * t3 * a6 &
```

Como é impossível uma avaliação 11 para as condições desta tabela (não existe nenhum valor para a que satisfaça as condições de ser menor que dois e maior que quatro), esta regra fica associada a uma transição vazia, quer dizer, a transição não indica nenhuma tabela para onde a rotina deve seguir. Neste caso, o resultado de uma avaliação jamais poderá determinar esta regra. Havendo a indicação desta regra durante o processamento da tabela, o resultado será imprevisível.

#### 5.6.2 - Uma tabela não pode ter regras incompatíveis

A avaliação das condições de uma tabela não pode determinar mais de uma regra que corresponda a conjuntos de ações ou transições diferentes. Não existe incompatibilidade entre duas regras se uma delas estiver associada a uma transição vazia.

### 5.6.3 - As regras podem ser redundantes

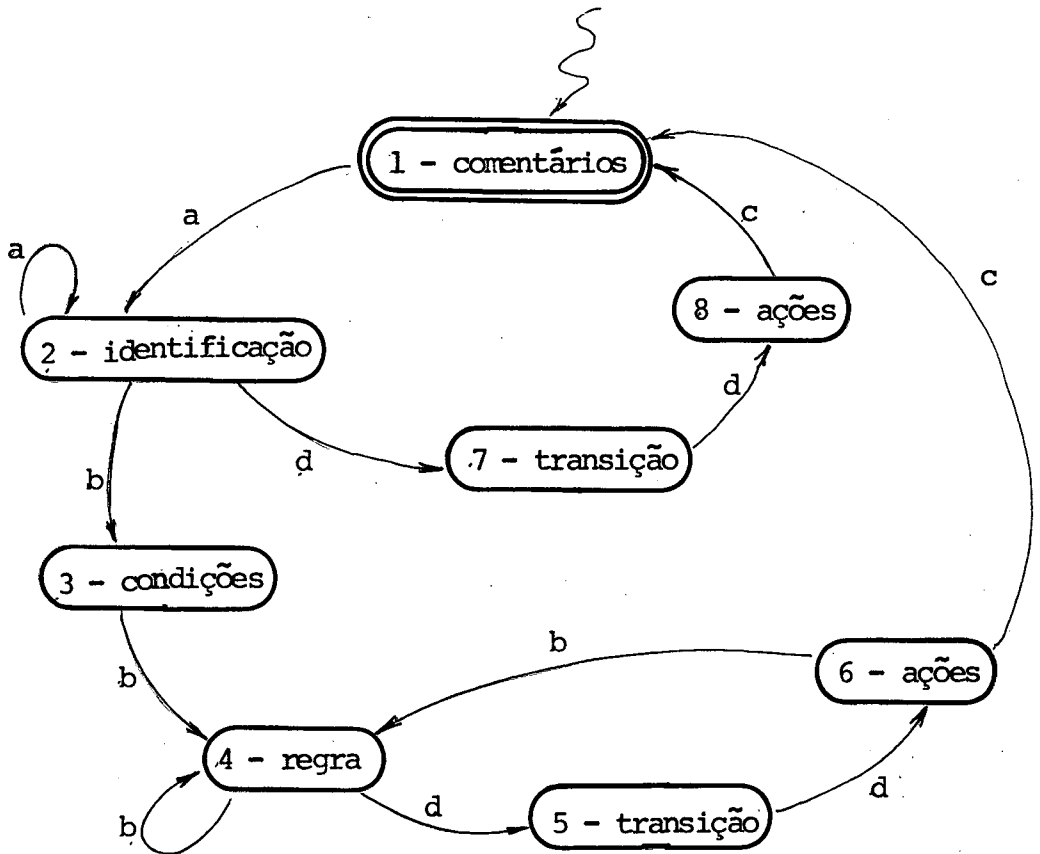
Uma avaliação de condições pode determinar mais de uma regra, desde que estas regras estejam associadas ao mesmo conjunto de ações e mesma transição.

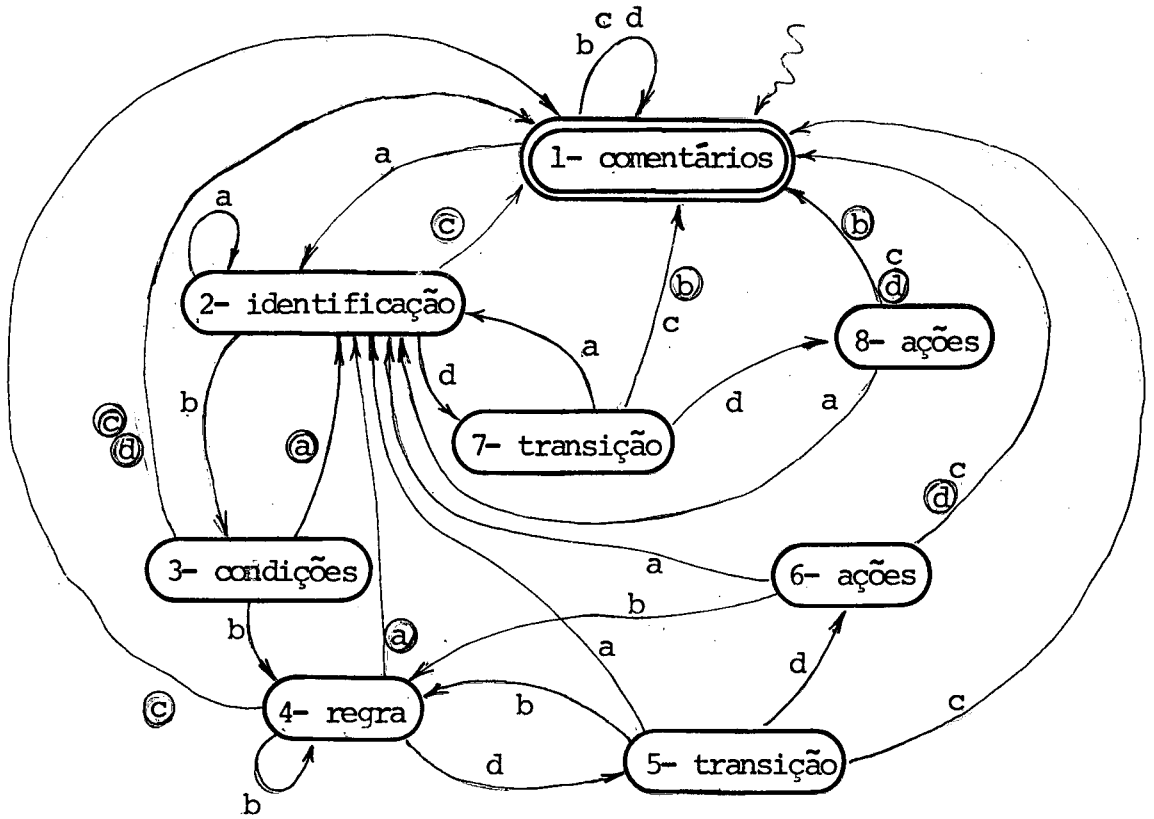
### 5.6.4 - A ordem das regras é irrelevante

As regras podem ser trocadas livremente, desde que continuem associadas aos conjuntos de ações e transições iniciais.

### 5.6.5 - A ordem das condições é irrelevante

As colunas de uma tabela podem ser trocadas sem alterar seu significado. Vale dizer, se forem trocadas as posições de duas condições e na sequência de símbolos de cada regra a ordem sofrer idêntica alteração, a tabela resultante é equivalente à anterior.

6 - Estrutura sintática da linguagem número 16.1 - Relação entre os componentes (primeiro nível):6.1.1 - Sintaxe-padrão (diagrama de transições)

6.1.2 - Sintaxe completa (diagrama de transições)

6.1.3 - Sintaxe-padrão (tabela de transições)

		símbolos de entrada				
		a	b	c	d	
e s t a d o s	1	2				comentários
	2	2	3		7	identificação
	3		4			condições
	4		4		5	regra
	5				6	transição
	6		4	1		ações
	7				8	transição
	8			1		ações

6.1.4 - Sintaxe completa (tabela de transições)

		símbolos de entrada			
		a	b	c	d
e s t a d o s	1	2	1	1	1
	2	2	3	①	7
	3	②	4	①	①
	4	②	4	①	5
	5	2	4	1	6
	6	2	4	1	①
	7	2	①	1	8
	8	2	①	1	①

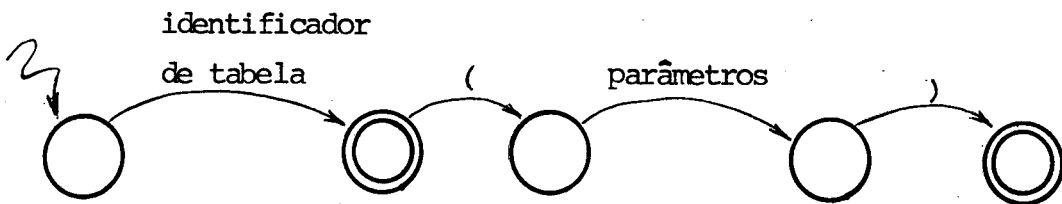
## 6.2 - Definição de componentes (segundo nível) :

### 6.2.1 - comentários

Qualquer sequência de caracteres não contendo símbolo a .

Não apresentam, igualmente, significado definido para área de controle.

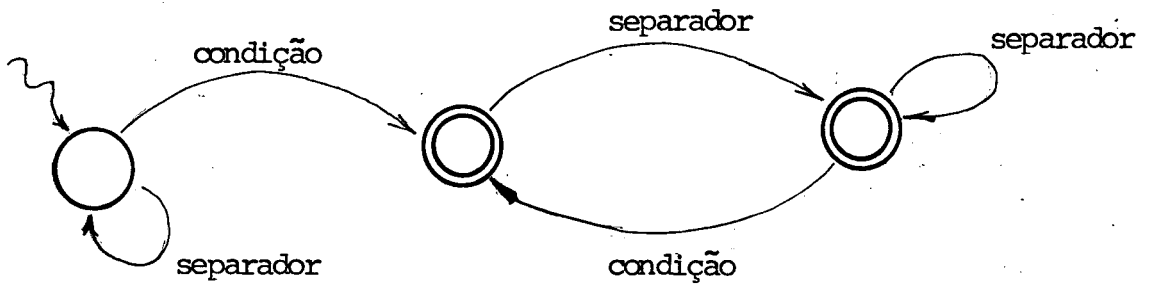
### 6.2.2 - identificação



### 6.2.3 - parâmetros

A ser definido na área de operações.

### 6.2.4 - condições

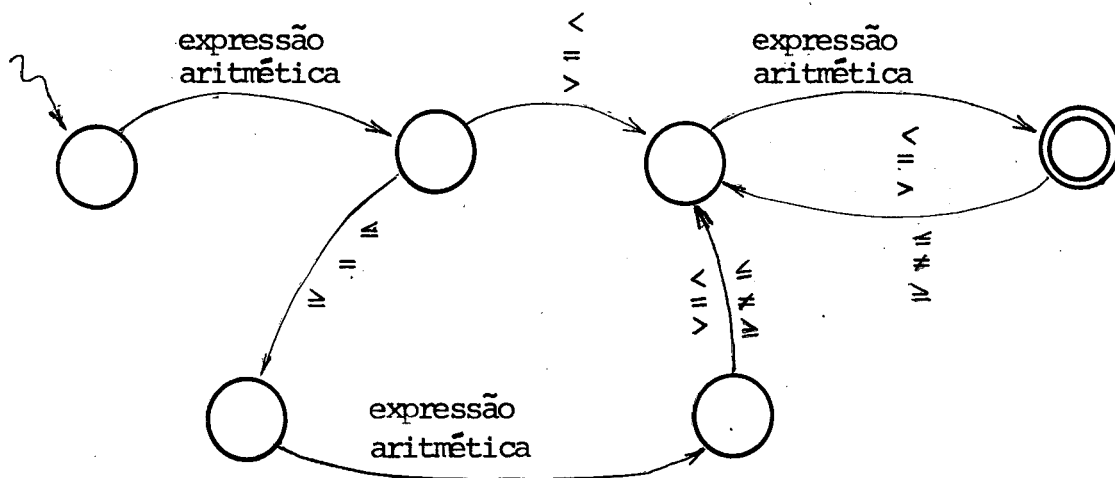


### 6.2.5 - condição

Qualquer afirmativa ou pergunta que possa ter como resposta sim ou não, não admitindo nenhuma outra. A sintaxe de condição deve ser definida junto com a sintaxe de ações (área de operações).

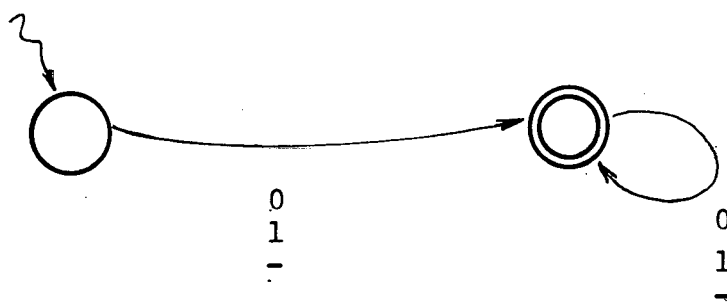
Será aqui apresentada, como sugestão inicial, um caso particular que trata de relações de expressões aritméticas.

### 6.2.6 - relação



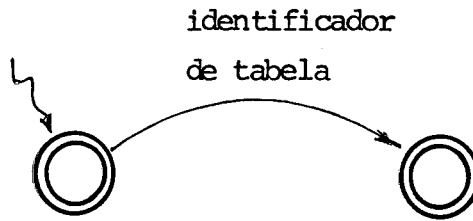
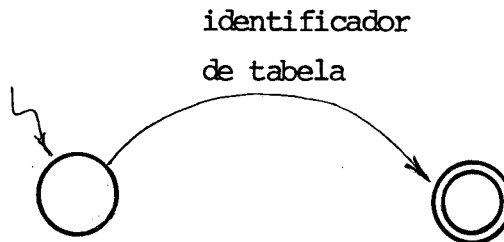
### 6.2.7 - regra

A quantidade de símbolos de uma regra tem que ser igual ao número de condições da mesma tabela.

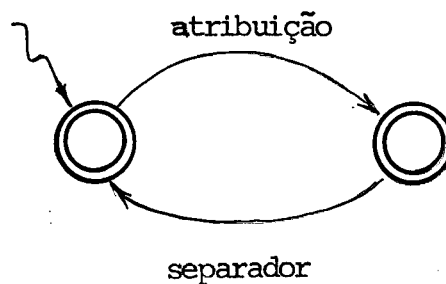


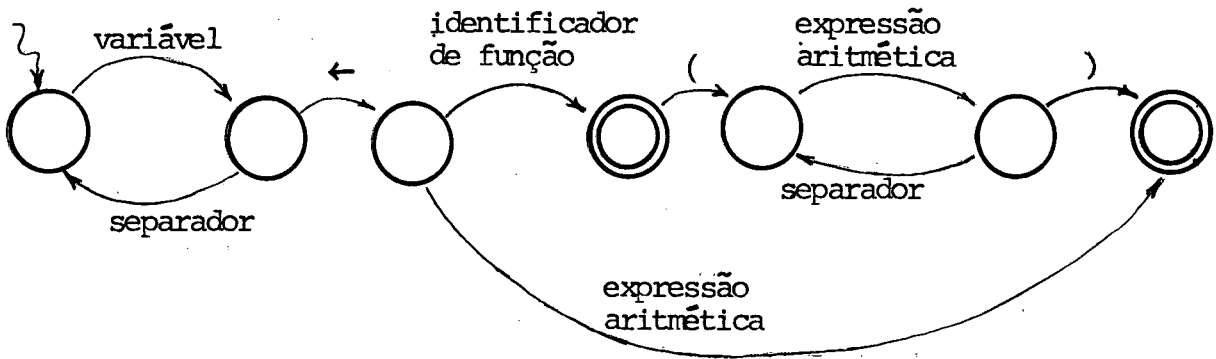
### 6.2.8 - transição

Existem duas sintaxes diferentes para transição : uma para transição em tabela com condições, outra para transição em tabela que não possui condições.

6.2.9 - transição em tabela com condições6.2.10 - transição em tabela sem condições6.2.11 - ações

Não está dentro do objetivo deste trabalho definir alguma sintaxe para ações, posto que deverá haver uma adequação entre aplicação e linguagem de operações. Mesmo assim, apresentaremos uma sugestão que pode ser utilizada em rotinas de processamento de dados. Nesta sintaxe, as entradas e saídas também são encaradas como atribuição.



6.2.12 - atribuição

6.3 - Exemplo (pesquisa em tabela)

Um tipo de rotina frequentemente usado em programação é a busca em tabela ou em arquivo, em que alguma sequência deve ser pesquisada. É muito comum esta pesquisa ter um tratamento semelhante ao da Figura 7, onde existe uma repetição que incrementa um índice ou atualiza um apontador e, a cada vez, testa se foi encontrada a chave desejada e se a tabela ou arquivo já foi esgotado. Mesmo quando a rotina abandona esta repetição, ainda é necessário fazer um novo teste para saber o que levou a abandoná-la. Se um compilador trabalhar com algum grau de otimização, esta rotina gera algo equivalente ao diagrama de blocos da Figura 8. Todavia, não é raro compiladores que geram rotinas equivalentes à Figura 9. A tradução da rotina da Figura 7 para a linguagem proposta é representada na Figura 10. Se, a partir do formato de tabelas, fosse necessário sequencializar os testes ali existentes, utilizaríamos um algoritmo para linearizar tabelas de decisões, chegariamos ao diagrama de blocos da Figura 11, que é menor e tem execução mais rápida que as duas opções apresentadas.

```
while i < n and a[i] ≠ x do i := i+1 ;
```

```
if a [i] = x then achou
      else não achou
```

Figura 7

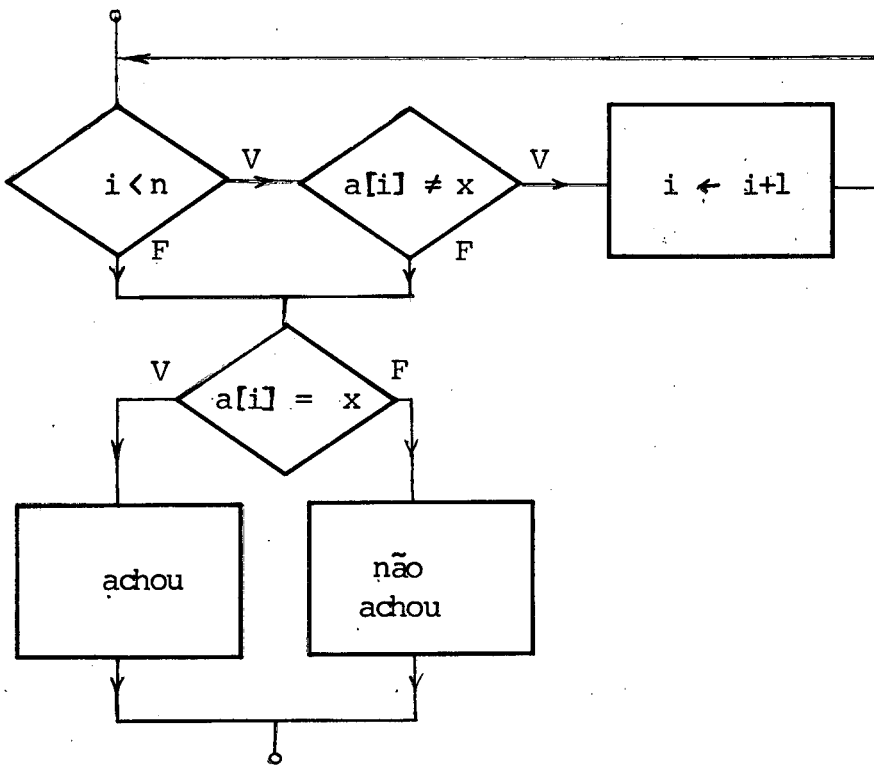


Figura 8

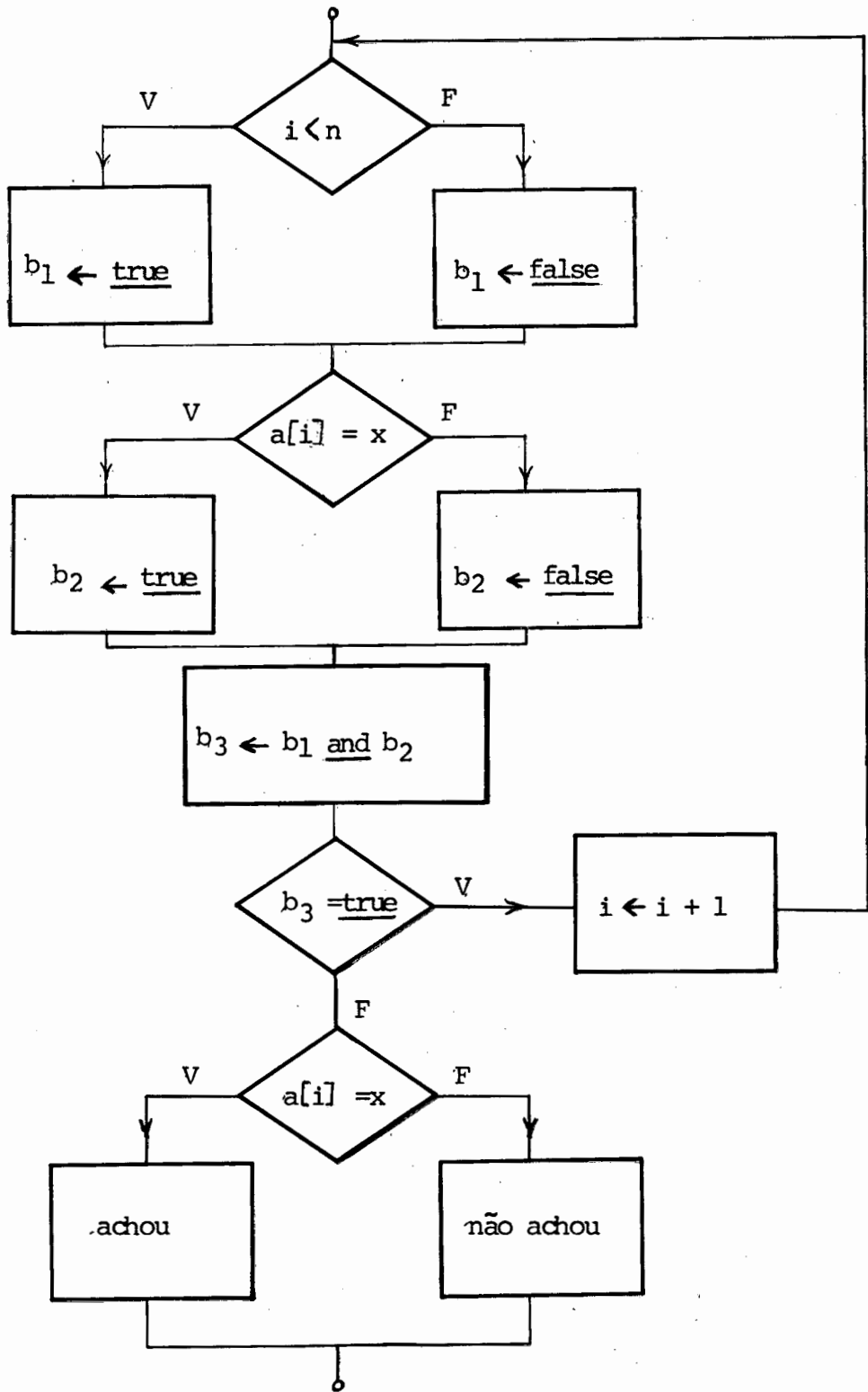


Figura 9

```

# t : a[i] = x . i < n
: 1 - * fim * achou
: 0 1 * t * i := i + 1
: 0 0 * fim * não achou &

```

Figura 10

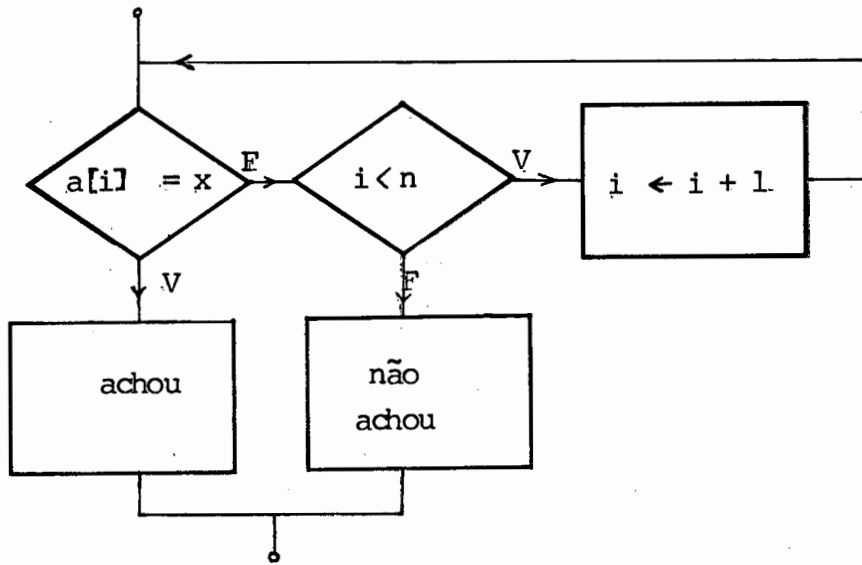


Figura 11

6.4 - Exemplo (greve dos metalúrgicos)

Em algum país, irrompera uma greve de metalúrgicos, que já se arrastava há trinta e quatro dias. Os trabalhadores reivindicaram: ou o executivo altera as leis vigentes (neste país não havia poder legislativo), ou se troca o ditador em exercício. Caso contrário, a greve continua. Os serviços de informações, ótimos analistas políticos (e sintetistas também), viram as seguintes hipóteses: ou se altera a lei, ou a greve continua (era época de abertura e não se queria usar o esquema policial militar). Avaliaram, também, o tempo máximo de duração da greve em um ano, pois os trabalhadores não dispunham de um fundo de greve suficiente para ultrapassar este período. O executivo declarou que "expludia" mas não mudaria a lei.

Chamemos as condições envolvidas de:

p - lei alterada,

q - greve existe,

r - "presidente" continua,

s - greve deflagrada há mais de um ano,

e usando a sintaxe definida, temos as posições de cada um:

```
# trabalhadores : p . q . r
                  : 1 - -
                  : - 1 -
                  : - - 0 * fim *
                  : 0 0 1 * * &
```

```
# informações : p . q . s
                : 1 - -
                : - 1 -
                : - - 1 * fim *
                : 0 0 0 * * &
```

```
# presidente : p
               : 0 * fim *
               : 1 * * &
```

Acrescentando o dado da realidade da greve e sua deflagração há menos de um ano, temos mais uma tabela.

```
# dados      : q . s
              : 1  0  * fim *
              : 0  -
              : -  1  *      *      &
```

Se unificarmos as tabelas: trabalhadores, informações, presidente e dados (neste caso, isto é possível), obteremos a tabela atual conjuntura.

```
# atual conjuntura : p . q . r . s
                  : 0  1  -  0  * fim *
                  : 1  -  -  -
                  : -  -  -  1
                  : -  0  -  -  *      *      &
```

O método empregado para a unificação das quatro tabelas iniciais em apenas uma final é bastante simples, pois que as regras que eram impossíveis nas tabelas iniciais permanecem impossíveis na final. Na tabela atual conjuntura a única regra possível é 01-0, que significa: a lei não é alterada, a greve continua e dura menos de um ano. Na tabela final, só aparece - (traço) na coluna referente à condição r, o que significa que esta coluna (e a condição correspondente) é irrelevante. Independe a renúncia ou permanência do presidente, uma vez que as posições iniciais continuarão inalteradas.

#### 6.5 - Exemplo (baseado em [KING 69])

Dada a Figura 12, confere-se o atendimento desta tabela às propriedades referentes à necessidade de uma tabela ser completa e à incompatibilidade entre as regras. Desta crítica resulta a Figura 13.

Esta segunda figura se constitui, inicialmente, da tabela da Figura 12. Logo a seguir, é apresentada a regra que denuncia a desobediência à primeira propriedade, ou seja, a tabela está incompleta. Como sinalização da sua não definição, a transição é representada pelo símbolo interrogação (?). Observando, agora, as condições desta tabela, sendo elas independentes, constatamos diversas incompatibilidades entre suas regras (Figura 13). Havendo, contrariamente, uma relação de dependência

entre as condições, a crítica realizada precisará de todas as informações que fazem as condições dependentes, sejam fornecidas por quem está descrevendo a rotina ou de alguma forma automática.

Partamos, então, para as especificações das condições da Figura 12 :  $c_1=(W_1 > r)$ ,  $c_2=(W_1 > 0)$ ,  $c_3=(W_1+W_2+W_3 > 0)$ ,  $c_4=(a > 3 \times r)$  e  $c_5=(a > 6 \times r)$ , para as variáveis  $W_1$ ,  $W_2$ ,  $W_3$ ,  $r$  e  $a$  positivas. Verificamos que estas condições não são independentes, havendo as seguintes implicações :

$$c_1 \rightarrow c_2, c_1 \rightarrow c_3, c_2 \rightarrow c_3 \text{ e } c_5 \rightarrow c_4.$$

Quando uma condição implica em outra, é equivalente dizer que é impossível a primeira ser avaliada como sim e, ao mesmo tempo, a segunda ser avaliada como não.

Esta equivalência é demonstrada na associação desta regra a uma transição vazia. A Figura 14 faz a correspondência das quatro regras impossíveis a transição vazia.

Desenvolvida na Figura 15, a tabela  $t$  com suas condições especificadas são acrescidas pelas regras impossíveis. Por um lado, vemos o fim da incompatibilidade mas, por outro, uma regra indefinida persiste.

#	$t$	:	$c_1$	.	$c_2$	.	$c_3$	.	$c_4$	.	$c_5$	
		:	1	-	-	-	-	0				
		:	-	0	1	0	-					
		:	0	1	-	0	-	*	$t_1$	*	$a_1$	
		:	1	-	-	-	1	*	$t_2$	*	$a_2$	
		:	0	1	-	1	0	*	$t_3$	*	$a_3$	
		:	-	-	0	-	0	*	$t_4$	*	$a_4$	
		:	-	-	0	-	1					
		:	-	0	1	-	1	*	$t_5$	*	$a_5$	
		:	-	0	1	1	0	*	$t_6$	*	$a_6$	&

Figura 12

# t	$c_1$	$c_2$	$c_3$	$c_4$	$c_5$	
: 1	-	-	-	0		
: -	0	1	0	-		
: 0	1	-	0	-		* t <sub>1</sub> * a <sub>1</sub>
: 1	-	-	-	1		* t <sub>2</sub> * a <sub>2</sub>
: 0	1	-	1	0		* t <sub>3</sub> * a <sub>3</sub>
: -	-	0	-	0		* t <sub>4</sub> * a <sub>4</sub>
: -	-	0	-	1		
: -	0	1	-	1		* t <sub>5</sub> * a <sub>5</sub>
: -	0	1	1	0		* t <sub>6</sub> * a <sub>6</sub>
: 0	1	1	1	1		* ? *
: 1	0	1	0	1		* t <sub>1</sub> * a <sub>1</sub>
						* t <sub>2</sub> * a <sub>2</sub>
						* t <sub>5</sub> * a <sub>5</sub>
: 1	-	0	-	0		
: -	1	0	0	0		* t <sub>1</sub> * a <sub>1</sub>
						* t <sub>4</sub> * a <sub>4</sub>
: 0	1	0	0	1		
: -	0	1	0	1		* t <sub>1</sub> * a <sub>1</sub>
						* t <sub>5</sub> * a <sub>5</sub>
: 1	0	1	1	0		* t <sub>1</sub> * a <sub>1</sub>
						* t <sub>6</sub> * a <sub>6</sub>
: 1	-	0	-	1		
: 1	0	1	-	1		* t <sub>2</sub> * a <sub>2</sub>
						* t <sub>5</sub> * a <sub>5</sub>
: 0	1	0	1	0		* t <sub>3</sub> * a <sub>3</sub>
						* t <sub>4</sub> * a <sub>4</sub>

&amp;

Figura 13

# impossíveis :  $c_1 \cdot c_2 \cdot c_3 \cdot c_4 \cdot c_5$

: 1 0 - - -

: 1 - 0 - -

: - 1 0 - -

: - - - 0 1 \* \* &

Figura 14

# t :  $W_1 > r$

.  $W_1 > 0$

. .  $W_1 + W_2 + W_3 > 0$

. . .  $a > 3 \times r$

. . . .  $a > 6 \times r$

. . . . .

: 1 - - - 0

: - 0 1 0 - \*  $t_1$  \*  $a_1$

: 1 - - - 1 \*  $t_2$  \*  $a_2$

: 0 1 - 1 0 \*  $t_3$  \*  $a_3$

: - - 0 - 0 \*  $t_4$  \*  $a_4$

: - - 0 - 1 \*  $t_5$  \*  $a_5$

: - 0 1 1 0 \*  $t_6$  \*  $a_6$

: 1 0 - - -

: 1 - 0 - -

: - 1 0 - -

: - - - 0 1 \* \*

: 0 1 - - 1 \* ? \* &

Figura 15

7 - Linguagem para tabelas de decisões com entrada mista7.1 - Entrada mista

Em tabelas de decisões com entrada mista, as condições têm que ser compostas antes de serem avaliadas. O símbolo... (reticências) pode aparecer inserido nas condições e será substituído por um complemento de condições referido nas regras. Feita esta substituição, serão avaliadas as condições. Caso surja a condição  $a...2$  e a regra correspondente for  $(=b+)$ , a relação que deve ser levada em consideração é  $a=b+2$ . Para efeito de comparação, introduzimos uma tabela de entrada mista e, em seguida, uma tabela equivalente, com entrada limitada.

```
# t : a...2
      : ( = ) * t1 * a1
      : ( < ) * t2 * a2
      : (=b+) * t3 * a3      &
```

Esta tabela, com apenas uma condição, equivale a uma tabela com entrada limitada que tenha três condições.

```
# t : a=2. a<2. a=b+2
      : 1   -   -   * t1 *a1
      : -   1   -   * t2 *a2
      : -   -   1   * t3 *a3
      : 1   1   -   *   *      &
```

Notamos que as duas tabelas com o mesmo significado estão incompletas e possuem regras incompatíveis. Detendo-nos no exemplo da tabela com entrada limitada, constatamos que nenhuma regra atende a uma avaliação das condições igual a 000 (as três condições avaliadas como não ). Por esta razão, ambas as tabelas estão incompletas, com a primeira e a terceira regra incompatíveis. Senão, vejamos: para uma avaliação igual a 111 ou 101 ou ainda 1-1 não está definido se deve ser escolhida a primeira ou a terceira regra. O mesmo se dá entre a segunda e a terceira regra, já que partindo da avaliação 111 ou 011 ou -11, existe uma indefinição na escolha entre uma ou outra.

## 7.2 - Estrutura sintática para tabelas de decisões com entrada mista

### 7.2.1 - Relação entre os componentes:

Para o emprego de tabelas de decisões com entrada mista pode ser utilizado o primeiro nível de qualquer das sintaxes.

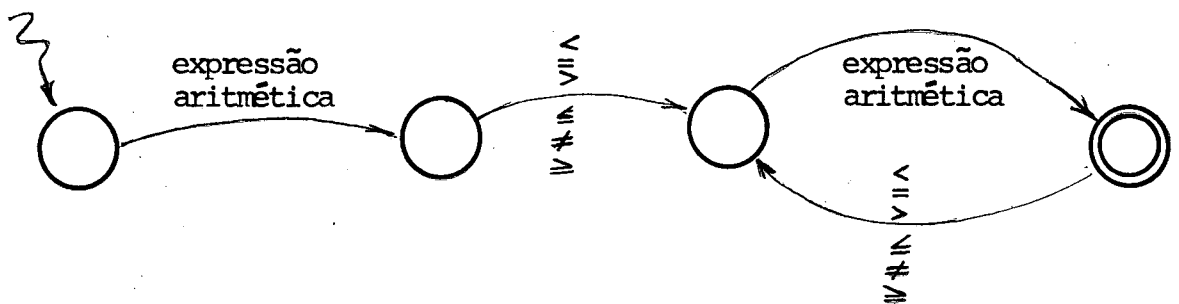
### 7.2.2 - Definição dos componentes:

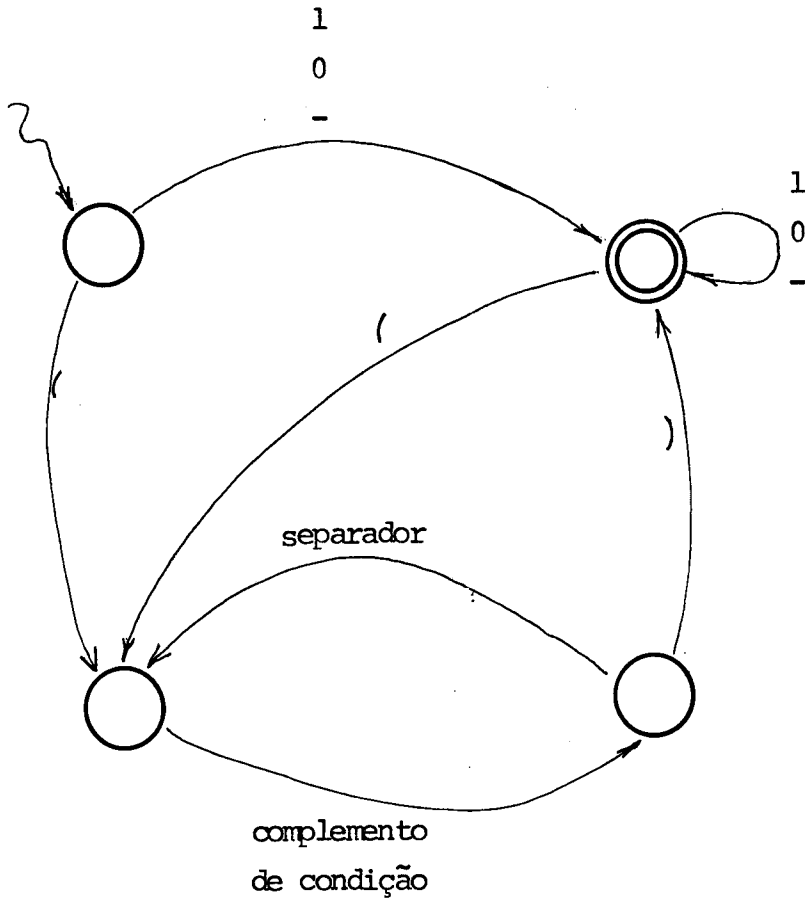
Dada a sintaxe básica, basta-nos alterar as definições de condição e de regra. Vejamos como se dá tal mudança:

#### 7.2.2.1 - Condição

Pode aparecer em qualquer lugar o símbolo reticências. Ele possibilitará sua substituição pelo correspondente complemento de condição que aparece em regra. Após esta ocorrência, retornamos à sintaxe de condição respeitando suas normas. Para o caso particular de relação, apresentaremos uma sugestão para tabelas com entrada mista.

#### 7.2.2.2 - relação



7.2.2.3 - regra

7.3 - Exemplo (baseado em[FISHER 66])

Na Figura 16, temos uma tabela na sintaxe proposta e, na Figura 17, estipulamos um trecho de programa em uma linguagem tipo Algol. Foi nossa preocupação fazer deste trecho uma conversão da tabela com os recursos que possibilitassem a redução de sua extensão.

```
# tab1 : function = ...
.      tr-code = valid
.      .      tr-empnbr = empnumber
.      .      .
: (act) - 1    * tab1 * move zero to act-code;
                read trans
: (adj) 1    1    * tab5 *
: (chg) 1    1    * tab6 *
: (del) - 1    * tab1 * do rodedit;
                write editrod to deleted;
                read trans
: (est) 1    -    * tab2 *
: (act) - 0
: (adj) 1    0
: (chg) 1    0
: (del) - 0    * tab1 * write empmast to newmast;
                read empmast
: (adj) 0    -
: (chg) 0    -
: (est) 0    -    * tab1 * write trans to error1;
                write errmsg to error1;
                read trans
```

&amp;

Figura 16

```

tab1 : if tr-code = valid
      then if function = est
          then go to tab2
          else if tr-empnbr = empnumber
              then if function = act
                  then go to a
                  else if function = adj
                      then go to tab5
                      else if function = chg
                          then go to tab6
                          else go to b

              else begin
                  write empmast to newmast;
                  read empmast;
                  go to tab1
                  end

          else if function = est or function = adj or function = chg
              then begin
                  write trans to error1;
                  write errmsg to error1 ;
                  go to tab1
                  end
              else if tr-empnbr ≠ empnumber
                  then begin
                      write empmast to newmast;
                      read empmast;
                      go to tab1
                      end
                  else if function = del
                      then b: begin
                          do rocredit;
                          write editrod to deleted;
                          read trans;
                          go to tab1
                          end
                      else a: begin
                          move zero to act-code;
                          read trans;

                          go to tab1
                          end

```

Figura 17

## 8 - Linguagem nº 2 - Linguagem reduzida

### 8.1 - Quantidade de símbolos

Embora apresentemos uma certa quantidade de símbolos nas sintaxes propostas, sabemos da capacidade de elas terem seus símbolos reduzidos sem perda de inteligibilidade. Um mesmo símbolo pode iniciar e dar fim a uma tabela. Tal como este exemplo, outras simplificações poderão ser feitas, se analisarmos mais detidamente caso por caso. Porém, como nossa intenção é diminuir a margem de enganos e a extensão de erros, optamos por uma linguagem redundante. Chegamos a propor sintaxes (por exemplo: número 7) com três símbolos diferentes precedendo ações e mais três para transição.

Com isto, sabe-se, além de que componente virá a seguir, quais são suas características e que tratamento terá, mesmo antes de ser lido.

A principal meta na apresentação desta sintaxe reduzida é o cotejo com a sintaxe básica. Tentaremos demonstrar que esta sintaxe poderia ser representada com metade dos símbolos propostos. Obviamente, esta colocação atingirá a formação da tabela, limitando em uma identificação para cada tabela.

8.2 - Relação entre os componentes8.2.1 - Sintaxe-padrão

	a	b	
1	2	7	comentários
2	3		identificação
3	4		condições
4	4	5	regra
5		6	transição
6	4	1	ações
7		8	transição
8		1	ações

8.2.2 - Sintaxe completa

	a	b	
1	2	1	comentários
2	3	7	identificação
3	4	①	condições
4	4	5	regra
5	4	6	transição
6	4	1	ações
7	②	8	transição
8	2	1	ações

## 9 - Linguagem número 3 - com preâmbulo

### 9.1 - Tabela com preâmbulo

Chamamos de tabela com preâmbulo uma tabela que possibilita ações serem executadas antes da avaliação das condições dessa tabela. No exemplo seguinte, o conjunto de ações  $a_1$  é executado e, somente após, as condições  $c_1$ ,  $c_2$  e  $c_3$  serão avaliadas.

# t		@ a <sub>1</sub>			
	:	c <sub>1</sub> · c <sub>2</sub> · c <sub>3</sub>			
	:	- 1 -			
	:	0 0 1 * t <sub>2</sub>	@ a <sub>3</sub>		
	:	1 0 1 * t <sub>4</sub>	@ a <sub>5</sub>		
	:	- 0 0 * t <sub>6</sub>	@ a <sub>7</sub>	&	

A tabela é equivalente às tabelas  $t'$  e  $t''$ , sendo que a única forma de ter acesso à tabela  $t''$  é através da tabela  $t'$ . Na tabela  $t$ , o preâmbulo se caracteriza pelo conjunto de ações  $a_1$ .

# t'		* t''	* a <sub>1</sub>	&
# t'':	c <sub>1</sub> · c <sub>2</sub> · c <sub>3</sub>			
	:	- 1 -		
	:	0 0 1 * t <sub>2</sub>	* a <sub>3</sub>	
	:	1 0 1 * t <sub>4</sub>	* a <sub>5</sub>	
	:	- 0 0 * t <sub>6</sub>	* a <sub>7</sub>	&

### 9.2 - Relação entre os componentes

#### 9.2.1 - Sintaxe-padrão (Figura x)

#### 9.2.2 - Sintaxe completa (Figura y)

sintaxe 7										
sintaxe 6										
sintaxe 5										
sintaxe 4										
sintaxe 3										
	a	b	c	d	e	f	g	h	i	
1	2									comentários
2	2	3		9	7	12	15	23	26	identificação
3		4								condições
4		4			5		13		24	regra
5				6		10		19		transição
6		4	1							ações
7				8		11		20		transição
8			1							ações
9		3								ações
10		4	1			10				ações paralelas
11			1			11				ações paralelas
12		3				12				ações paralelas
13				14		17		21		transição paralela
14		4	1				13			ações
15				16		18		22		transição paralela
16			1				15			ações
17		4	1			17	13			ações paralelas
18			1			18	15			ações paralelas
19		4	1					19		ações não-determinísticas
20			1					20		ações não-determinísticas
21		4	1				13	21		ações não-determinísticas
22			1				15	22		ações não-determinísticas
23		4						23		ações não-determinísticas
24				25		28		30		transição não-determinística
25		4	1						24	ações
26				27		29		31		transição não-determinística
27			1						26	ações
28		4	1			28			24	ações paralelas
29			1			29			26	ações paralelas
30		4	1					30	24	ações não-determinísticas
31			1					31	26	ações não-determinísticas

sintaxe 7

sintaxe 6

sintaxe 5

sintaxe 4

sintaxe 3

	a	b	c	d	e	f	g	h	i
1	2	1	1	1	1	1	1	1	1
2	2	3	①	9	7	12	15	23	26
3	②	4	①	①	①	①	①	①	①
4	2	①	①	①	5	①	13	①	24
5	2	4	1	6	①	10	①	19	①
6	2	4	1	①	①	①	①	①	①
7	2	①	1	8	①	11	①	20	①
8	2	①	1	①	①	①	①	①	①
9	②	3	①	①	①	①	①	①	①
10	2	4	1	①	①	①	①	①	①
11	2	①	1	①	①	11	①	①	①
12	②	3	①	①	①	①	①	①	①
13	2	4	1	14	①	17	13	21	①
14	2	4	1	①	①	①	13	①	①
15	2	①	1	16	①	18	15	22	①
16	2	①	1	①	①	①	15	①	①
17	2	4	1	①	①	17	13	①	①
18	2	①	1	①	①	18	15	①	①
19	2	4	1	①	①	①	①	19	①
20	2	①	1	①	①	①	①	20	①
21	2	4	1	①	①	①	13	21	①
22	2	①	1	①	①	①	15	22	①
23	②	3	①	①	①	①	①	23	①
24	2	4	1	25	①	28	①	30	24
25	2	4	1	①	①	①	①	①	24
26	2	①	1	27	①	29	①	31	26
27	2	①	1	①	①	①	①	①	26
28	2	4	1	①	①	28	①	①	24
29	2	①	1	①	①	29	①	①	26
30	2	4	1	①	①	①	①	30	24
31	2	①	1	①	①	①	①	31	26

Figura y

comentários  
 identificação  
 condições  
 regra  
 transição  
 ações  
 transição  
 ações  
 ações  
 ações paralelas  
 ações paralelas  
 ações paralelas  
 transição paralela  
 ações  
 transição paralela  
 ações  
 ações paralelas  
 ações paralelas  
 ações não-determinísticas  
 ações não-determinísticas  
 ações não-determinísticas  
 ações não-determinísticas  
 ações não-determinísticas  
 transição não-determinística  
 ações  
 transição não-determinística  
 ações  
 ações paralelas  
 ações paralelas  
 ações não-determinísticas  
 ações não-determinísticas

### 9.3 - Exemplo

Na Figura 18, um programa lê os três lados de um triângulo e determina o seu tipo. Este é um programa essencialmente combinacional, porém resolvido linearmente, já que a linguagem disponível era linear.

Com este exemplo de programa, faremos a tradução para a sintaxe proposta, juntamente a uma série de alterações, mas sempre mantendo o algoritmo original.

A Figura 19 é a realização desta tradução.

Como a variável m não influi nas avaliações da tabela  $t_1$  (Figura 19), a ação  $m := 0$  pode ser deslocada para depois das condições desta tabela. O conjunto de ações referentes à primeira regra resulta em  $m := 0; m := m + 1$ , que pode ser substituído por  $m := 1$ , enquanto o conjunto associado à segunda regra é  $m := 0$  (Figura 20). Porque a ação read (i, j, k) altera a avaliação das condições desta tabela (altera os valores de i e j), não pode ser deslocada do preâmbulo.

As transições da nova tabela  $t_1$  (Figura 20) são dirigidas todas para  $t_2$  e ficam associadas a ações que não alteram a avaliação das condições de  $t_2$ . Estas duas tabelas podem ser unificadas, gerando a tabela  $t_1$  da Figura 21. Da mesma forma, esta nova tabela não traz consigo ações que possam influir na avaliação das condições da tabela  $t_3$ . As suas transições direcionam-se todas, e somente elas na rotina, para  $t_3$ . A Figura 22 unifica a tabela  $t_3$  com a  $t_1$  da Figura 21. A tabela gerada por esta ligação apresenta uma característica peculiar no caso das regras. Se duas condições quaisquer forem avaliadas como sim, conseqüentemente a terceira também optará pelo sim. Nesta medida, constatamos a impossibilidade de existência das regras 110, 101 ou 011.

Ainda podemos fazer nesta rotina mais uma transformação. A tabela  $t_1$  com sua nova forma após as alterações, aponta todas suas transições para a tabela  $t_4$  e suas ações influenciam na avaliação das condições desta tabela. Mas, cada ação de  $t_1$  (Figura 22) determina um valor para m e, na tabela  $t_4$ , as condições que levam em consideração m testam qual seu valor. As avaliações da tabela  $t_4$  em relação a m são desnecessárias, porque no julgamento das condições de  $t_1$ , já fica determina

do o valor de  $\underline{m}$ . Com isto, pode-se unificar as tabelas  $t_1$  e  $t_4$ , eliminando as condições que avaliam  $\underline{m}$  e as atribuições de valores a  $\underline{m}$ .

Na Figura 23 a rotina se completa depois destas transformações. Esta rotina final é equivalente à inicial (Figura 18), em que o algoritmo utilizado foi preservado.

```
read ( i, j, k ) ;
```

```
m := 0 ;
```

```
if i=j then m := m+1 ;
```

```
if i=k then m := m+2 ;
```

```
if j=k then m := m+3 ;
```

```
if m=0
```

```
  then if i+j <= k or j+k <= i or i+k <= j
```

```
    then não triângulo
```

```
    else escaleno
```

```
  else if m=1
```

```
    then if i+j = k
```

```
      then não triângulo
```

```
      else isósceles
```

```
    else if m=2
```

```
      then if i+k <= j
```

```
        then não triângulo
```

```
        else isósceles
```

```
    else if m=3
```

```
      then if j+k <= i
```

```
        then não triângulo
```

```
        else isósceles
```

```
      else equilátero
```

Figura 18

```

# t1          @ read (i, j, k) ;
                m := 0
                : i=j
                : 1 * t2 @ . m := m+1
                : 0 * t2 @           &

# t2 : i=k
                : 1 * t3 @ m := m+2
                : 0 * t3 @           &

# t3 : j=k
                : 1 * t4 @ m := m+3
                : 0 * t4 @           &

# t4 : m=0 . m=1 . m=2 . m=3 . i+j>k . j+k>i . i+k>j
                : 1 - - - 0 - -
                : 1 - - - - 0 -
                : 1 - - - - - 0
                : 0 1 - - 0 - -
                : 0 0 1 - - - 0
                : 0 0 0 1 - 0 - *fim@ não triângulo
                : 1 - - - 1 1 1 *fim@ escaleno
                : 0 1 - - 1 - -
                : 0 0 1 - - - 1
                : 0 0 0 1 - 1 - *fim @ isósceles
                : 0 0 0 0 - - - *fim @ equilátero &

```

Figura 19

```

# t1                @    read ( i, j, k)
: i=j
: 1      * t2 @ m := 1
: 0      * t2 @ m := 0          &

```

Figura 20

```

# t1                @    read (i, j, k)
: i=j . i=k
: 1      1      * t3 @ m := 3
: 1      0      * t3 @ m := 1
: 0      1      * t3 @ m := 2
: 0      0      * t3 @ m := 0          &

```

Figura 21

```

# t1                @    read ( i, j, k)
: i=j . i=k . j=k
: 1      1      1      * t4 @ m := 6
: 1      0      0      * t4 @ m := 1
: 0      1      0      * t4 @ m := 2
: 0      0      1      * t4 @ m := 3
: 0      0      0      * t4 @ m := 0
: 1      1      0
: 1      0      1
: 0      1      1      *      @          &

```

Figura 22

```

# t1                                     @ read (i,j,k)

: i=j . i≠k . j≠k . i+j>k . i+k>j . j+k>i

: 1  1  1  -  -  -      * fim @ equilátero
: -  0  0  0  -  -
: 0  -  0  -  0  -
: 0  0  -  -  -  0      * fim @ não triângulo
: 1  0  0  1  -  -
: 0  1  0  -  1  -
: 0  0  1  -  -  1      * fim @ isósceles
: 0  0  0  1  1  1      * fim @ escaleno
: 0  1  1  -  -  -
: 1  0  1  -  -  -
: 1  1  0  -  -  -      * e &

```

Figura 23

## 10 - Linguagem número 4 - com ações paralelas

### 10.1 - Ações paralelas

Nas rotinas em que se aceitam ações paralelas, pode ocorrer a associação de uma regra a mais de um conjunto de ações, em uma ou mais tabelas. Por exemplo:

```
# t : c
    : 1 * u @ a1
    : 0 * v % a2
           % a3
           % a4           &
```

Quando a avaliação de c for 0 (não), teremos esta regra associada a três conjuntos de ações: a<sub>2</sub>, a<sub>3</sub> e a<sub>4</sub>. Neste caso, as execuções dos conjuntos de ações são independentes, podendo ser realizadas em paralelo, sem qualquer sincronismo interno. Só após a conclusão das três ações se rá feita a transição para v.

### 10.2 - Relação entre os componentes:

#### 10.2.1 - Sintaxe-padrão (Figura x)

#### 10.2.2 - Sintaxe completa (Figura y)

## 11 - Linguagem número 5 - com transições paralelas

### 11.1 - Transições paralelas

Para que existam transições paralelas, é necessário levar em consideração dois aspectos: bifurcação e junção. Bifurcação é o ponto em que uma rotina se divide em dois ou mais ramos, a partir do qual todos eles podem ser seguidos simultaneamente. Junção é o ponto onde dois ou mais ramos da rotina se unem.

#### 11.1.1 - Bifurcação

Existem duas formas de indicar bifurcação na sintaxe proposta:

- 1 - Para uma mesma regra corresponde mais de um conjunto de ações e mais de uma transição. Exemplo:

```
# p : c1 . c2
      : 1 0 * q @ a1
      : 1 1 ! r @ a2
              ! s @ a3
      : 0 - * t @ a4 &
```

Se  $c_1$  e  $c_2$  forem avaliados como 1 e 1, a rotina segue por dois caminhos:

- a) executa  $a_2$  e segue para r,
- b) executa  $a_3$  e segue para s.

Estes caminhos podem ser seguidos simultaneamente, sem nenhuma sincronização.

- 2 - A uma mesma regra corresponde um conjunto de ações e mais de uma transição. Exemplo:

```
# r : c1 . c2
      : 1 0 * q @ a1
      : 1 1 ! u s @ a2
      : 0 - * t @ a3 &
```

Esta construção é equivalente ao seguinte trecho de rotina:

```
# r : c1 . c2
    : 1 0 * q @ a1
    : 1 1 * r' @ a2
    : 0 - * t @ a3 &

# r' : u @
     : s @ &
```

Neste caso, somente a tabela r pode alcançar a tabela r' .

### 11.1.2 Junção

Para que seja possível fazer o sincronismo entre diversos caminhos de uma rotina, uma tabela só será processada quando um determinado número de transições fizer referência a ela. Exemplo:

```
# (2) teste : c1 . c2
            : 1 -
            : - 1 *t1 @ a2
            : 0 0 *t2 @ a1 &
```

A tabela teste só será processada quando aparecer na rotina duas transições que tentem atingi-la.

Em uma tabela podem existir transições paralelas sinalizando para uma mesma tabela. Neste caso, há duas maneiras de indicação; nos exemplos a seguir, as duas tabelas são equivalentes.

```
# tab : c
      : 1 ! t t @ a1
      : 0 * tab @ a2 &
```

```
# tab : c
      : 1 ! (2) t @ a1
      : 0 * tab @ a2 &
```

## 11.2 - Estrutura sintática da linguagem número 5

### 11.2.1 - Relação entre componentes:

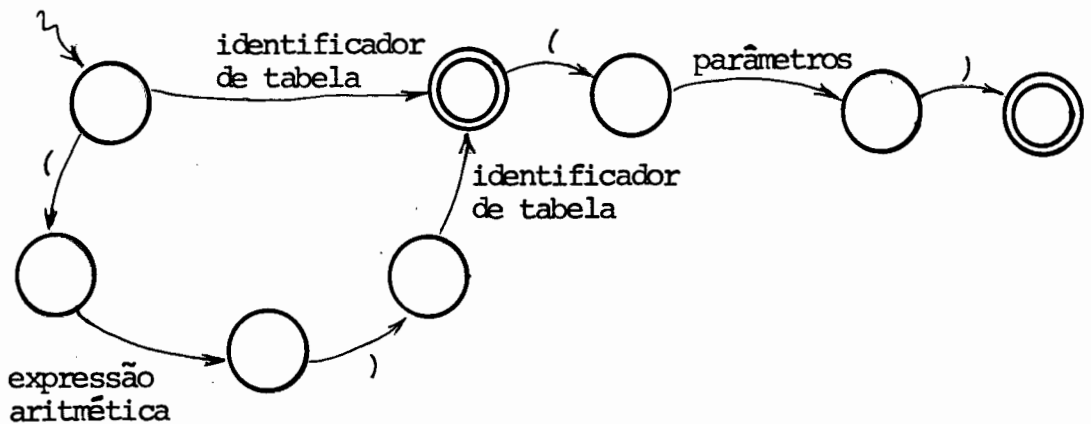
#### 11.2.1.1 - Sintaxe-padrão (Figura x)

#### 11.2.1.2 - Sintaxe completa (Figura y)

### 11.2.2 - Definição de componentes:

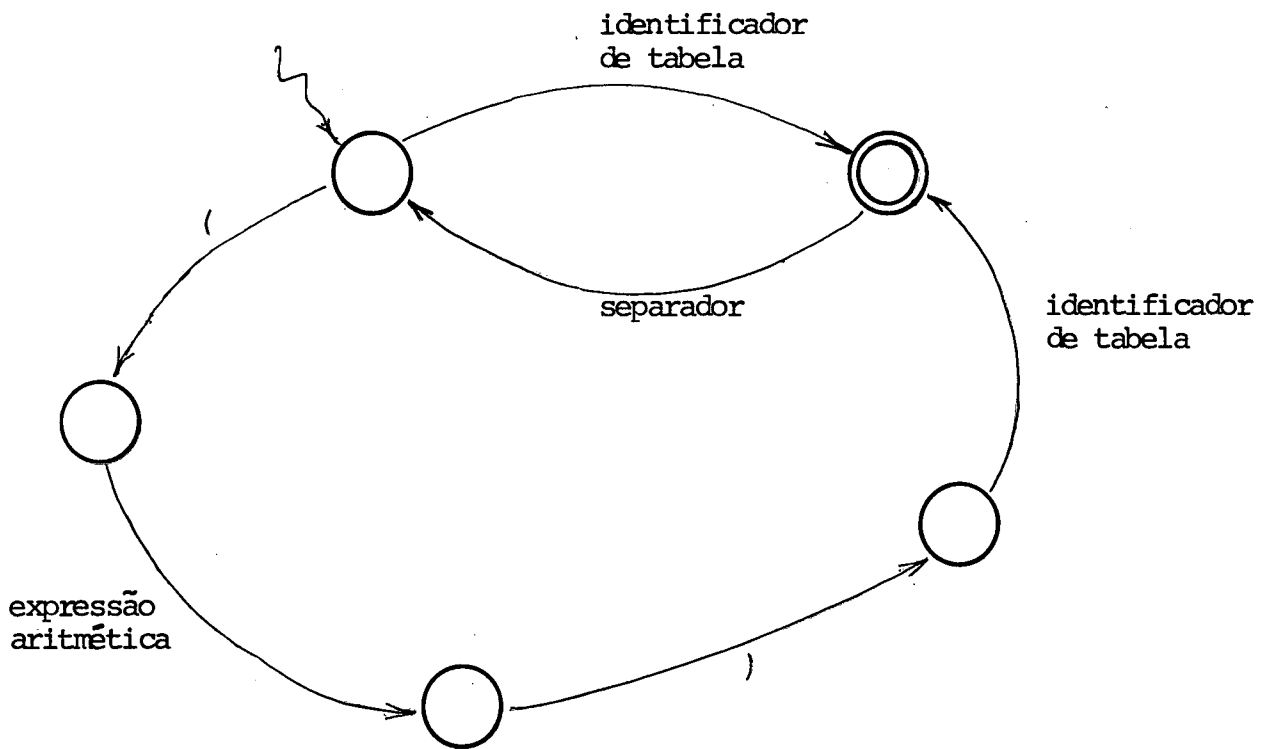
Serão alterados apenas os conceitos de identificação e transição.

#### 11.2.2.1 identificação



#### 11.2.2.2 - transição

Existem duas formas de transição: uma onde não existe paralelismo (transição precedida pelo símbolo e) e outra a partir da qual pode existir mais de um caminho (transição paralela, precedida pelo símbolo g).

11.2.2.3 Transição paralela

## 12 - Linguagem número 6 - com ações não-determinísticas

### 12.1 - Não-determinismo

O conceito de não-determinismo, para nosso trabalho, é a opção que uma rotina tem na escolha de um caminho entre vários outros, com a obrigatoriedade de chegar ao fim.

Este conceito é aplicado efetivamente nesta linguagem de número 6, bem como na linguagem número 7.

### 12.2 - Ações não-determinísticas

Uma regra pode estar associada a mais de um conjunto de ações, de forma que não deterministicamente tenha que escolher uma delas. Exemplo:

# t	:	c				
	:	1	*	t <sub>1</sub>	\$	a <sub>1</sub>
					\$	a <sub>2</sub>
					\$	a <sub>3</sub>
	:	0	*	t <sub>2</sub>	@	a <sub>4</sub>
						&

Nesta tabela, se c for avaliada como 1 (sim), a escolha deve recair em um dos três conjuntos de ações associados à primeira regra. Neste ponto, uma escolha acertada é qualquer uma que possibilite, com o conjunto de ações escolhido, alcançar o final da rotina da qual esta tabela faz parte. Após escolhido e executado o conjunto de ações, a rotina se que para t<sub>1</sub> .

### 12.3 - Relação entre os componentes:

#### 12.3.1 - Sintaxe-padrão (Figura x)

#### 12.3.2 - Sintaxe completa (Figura y)

### 13 - Linguagem número 7 - com transições não-determinísticas

#### 13.1 - Transições não-determinísticas

Associação de uma mesma regra a várias transições.

Quando tal ocorrer, a escolha deve recair, não-deterministicamente, sobre uma dessas transições. Exemplo:

```
# t : c
      : 1   * t1 @ a1
      : 0   ? t2 @ a2
           ? t3 @ a3
           ? t4 @ a4           &
```

Se, nesta tabela, c tomar o valor 0 (não), existem três possibilidades de continuação:

- ou é executado a<sub>2</sub> e segue para t<sub>2</sub>,
- ou é executado a<sub>3</sub> e segue para t<sub>3</sub>,
- ou é executado a<sub>4</sub> e segue para t<sub>4</sub>.

O critério para a escolha de um destes caminhos é o mesmo de sempre, deve ser escolhido algum que leve ao final da rotina.

No caso de rotinas que aceitam transições não-determinísticas, continua a existir a obrigatoriedade de as tabelas serem completas. Todavia, pode-se indicar para uma determinada regra, que há um caminho que não deve ser seguido. Para se objetivar esta situação, associa-se a uma regra uma transição não-determinística que não indica nenhuma tabela. Exemplo:

```
# t : c
      : 1   * t1 @ a1
      : 0   ?   @           &
```

Neste caso, se c for avaliada como 0 (não), é sinal que em algum ponto da rotina foi tomado um caminho errado, pois não existe possibilidade, com esta avaliação, de se chegar ao final da rotina.

## 13.2 - Estrutura sintática da linguagem número 7

### 13.2.1 - Relação entre os componentes:

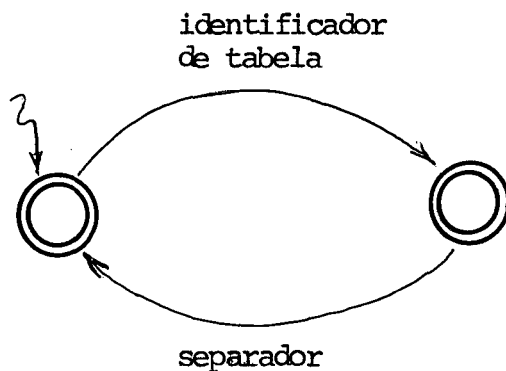
#### 13.2.1.1 - Sintaxe-padrão (Figura x)

#### 13.2.1.2 - Sintaxe completa (Figura y)

### 13.2.2 - Definição dos componentes :

Existe, neste caso, mais um tipo de transição, que é a transição não-determinística. Esta transição aparece sempre após o símbolo i.

#### 13.2.2.1 - transição não-determinística



14 - CONCLUSÕES

Retomamos tabelas de decisões, aqui com formato livre, onde abordamos isoladamente a área de controle.

As linguagens lineares, obrigando um rebaixamento de dimensões, tornam-se demasiado ambíguas, além do grande acréscimo de símbolos exigidos. Um exemplo desta ambiguidade é o uso do símbolo do. Não se consegue evidenciar quando e de que forma deve ser avaliada a condição envolvida, que valores tomam as variáveis e se podem ser alteradas ou não. As discussões persistem há mais de duas décadas e não tendem para um consenso.

As linguagens aqui desenvolvidas, por serem não lineares, descrevem aspectos não lineares de uma forma mais adequada. Tornam-se mais fáceis o aprendizado e emprego de uma linguagem com estas características, bem como fazer alterações e compreender uma rotina. Tal linguagem é mais potente e tão ou mais simples que diagrama de blocos.

Com as linguagens propostas existe uma real convergência no sentido de um ato único cobrindo duas atividades:

programar (comunicação homem-máquina) e documentar (comunicação entre pessoas). Nosso trabalho foi desenvolvido com a preocupação de perseguir o conceito: documentar é programar. Para isto, cremos que o caminho é introduzir maior formalismo na documentação e permitir que as pessoas lidem com linguagens de programação simplificadas.

Outro benefício alcançado é a introdução de um mínimo de informações irrelevantes à solução de um determinado problema. Deve-se procurar linguagens onde seja possível transmitir informações apenas inerentes à solução do problema e não para atender ao meio de comunicação disponível.

Quanto menos informações irrelevantes houver, mais fácil fazer alterações na forma de uma rotina sem alterar seu significado. Os problemas na diminuição do tempo de execução ou do espaço ocupado por uma rotina (otimização) ficam bastante reduzidos. As linguagens propostas possibilitam otimizações mais adequadas às máquinas, mesmo quando estas

operam sequencialmente.

A análise semântica fica também beneficiada com a minimização de informações não pertinentes. Entre outros fatores, se torna mais simples constatar em uma tabela se todas as possíveis avaliações das condições estão presentes, árdua tarefa em uma linguagem linear. Além de ser possível aplicar todas as operações lógicas às condições (desde que transformadas para a forma sintática adequada), existe um modo de explicitar quais as relações lógicas, sendo às vezes exigida a apresentação destas relações para tomar completa uma tabela.

Com a participação de uma linguagem em suas três áreas básicas - especificações, operações e controle - não existe mais a necessidade de um potente compilador e sim de três compiladores bastante simplificados. A área de controle é representada por uma linguagem tão regular quanto as linguagens atuais são livre de contexto. Em verdade, para a análise desta área é necessário um autômato finito e um referencial com os identificadores das tabelas e suas características.

Com este esquema de linguagem surge, de imediato, o questionamento do atual meio ambiente que envolve um programa (ou rotina). Parece bastante natural que se repense a estrutura dos sistemas operacionais e a arquitetura das máquinas, sendo possível criar máquinas mais especializadas e, portanto, mais simples, tratando mais eficientemente cada área. A área de especificações merece uma arquitetura própria, que talvez surja dos estudos de Dicionários de Dados/Diretórios e Bancos de Dados. Quanto à área de operações, esta vê surgir uma arquitetura a partir das máquinas que operam com fluxos de dados.

À parte de controle se faz necessária uma lógica combinacional para, a partir da avaliação das condições de uma tabela, determinar quais ações devem ser executadas e qual a próxima tabela. Um subsistema ALPC (Arranjo Lógico Programável Combinacional - ver [ZUFFO 77]), que não possui nenhum bit estável interno, vale dizer, não possui memória, pode processar uma tabela, admitindo paralelismo ou não-determinismo. Uma tabela armazenada como matriz booleana, pode vir a ser o micro-programa de um ALPC micro-programável ou algo equivalente. Empregada esta arquitetura, uma tabela é avaliada em um passo.

Este trabalho, evidentemente, não esgota a área de controle, posto que muito precisa ser visto ainda, contudo, intentamos ampliar o âmbito das discussões existentes e estamos abertos às críticas e sugestões.



---

Carlos Flores Cunha

Rua Ronald de Carvalho 292 / 902

Rio de Janeiro

Tel: 257-2336

22021

Bibliografia

Estamos dividindo a bibliografia em dois blocos, um é o levantamento bibliográfico sobre tabelas de decisões, o outro é constituído das referências feitas no texto a publicações que não tratam de tabelas de decisões.

Referências que não tratam de tabelas de decisões

- [KNUTH 74] Knuth, Donald E. - Structured Programming with go to Statements. Computing Surveys 6, 4 (dez 1974) 261-301.
- [WEGNER 78] Wegner, Peter - Supporting a Flourishing Language Culture. SIGPLAN Notices 13, 12 (dez 1978), 102-104.
- [ZUFFO 77] Zuffo, João Antonio - Circuitos Integrados em média escala e em larga escala. Edgard Blücher, São Paulo (1977).

ANEXO BLevantamento bibliográfico sobre tabelas de decisões

- 'AINSLE & KENNEY 72' AINSLE, R.J. KENNEY, A.A. - A Tool for tax Practitioners. The Tax Advisor (jun 1972), 336-345.
- 'AIR FORCE' AIR FORCE, U.S. - Decision Tables for Regulations. Pamphlet 5-1-1, Chapter 3.
- 'ARMERDING 62' ARMERDING, G.W. - FORTAB : A Decision Table Language for Scientific Computing Applications. Rand Corp., Santa Monica, California, Set 1962.
- 'ARNOLD & HALBACH 75' ARNOLD, G. HALBACH, H.D. - Neue Aspekte zur Entscheidungstabellentechnik. Elektronische Rechenanlagen mit Computer Praxis (Munche)n 17, 6 (dez 1975), 282-286.
- 'ARNOLD 71' ARNOLD, H.O. - Utilization of a Decision Table Translator for BASIC Program Creation . SIGPLAN Notices 6: No. 8:12-19 (set 1971).
- 'AUERBACH 68' AUERBACH - Decision Tables - Their General Construction and Acceptance in Programming. Auerbach Standard EDP Reports, (1968), pp. 23 030 100-103.
- 'BAGLIN & KLEE 73' BAGLIN, G. KLEE, J. - Las Tablas de Decision, Instrumento para el Analisis Informativo. Bilbao, Deusto, 1973. 129 P. (Coleccion Informatica).
- 'BARALT-TORRIJOS & PASS 69' BARALT-TORRIJOS J. PASS, E.M. - Decision Table Minimization. GITIS 69-23. Georgia Inst. Technol., Atlanta, Georgia, 1969.

- 'BARGULYA 76' BARGULYA, I. - Programming of Decision Tables in EMG ALGOL 60 Language. Inf. Elektron. (Hungary) 11, 3 (1976), 225-229.
- 'BARNARD 67' BARNARD, T.J. - NITA User's Guide . NCC Newsletter. Nat. Comput. Centre, Manchester, England, 1967.
- 'BARNARD 69' BARNARD, T.J. - A New Rule Mask Technique for Interpreting Decision Tables. Comput. Bull. 13: 153-154 (maio 1969).
- 'BAYES 73' BAYES, A.J. - A Dynamic Programming Algorithm to Optimise Decision Table Code. Australian Computer J. 5,2 (maio 1973), 77-79.
- 'BJORK 68' BJORK, HARRY - Decision Tables in ALGOL 60. BIT, 8 (1968), 147-153.
- 'BOERDAM 66a' BOERDAM, W. et al - The DETAB/65 Language, DO 3396. SHARE General Program Library, jan. 1966.
- 'BOERDAM 66 b' BOERDAM, W. et al - DETAB/65 User's Manual, DO 3396. SHARE General Program Library, jan 1966.
- 'BROWN 62' BROWN, L.M. - Decision Table Experience on a File Maintenance System . Proc. Decision Tables Symp. 75-80. Assoc. Comput. Machinery, New York, 1962.
- 'BUCKERFIELD 69' BUCKERFIELD, P.S.T. - A Technique for the Construction and Use of a Generalised Information Table . Proc. IFIP Congr. 68.395-402. North-Holland Publ., Amsterdam, 1969.
- 'C.E.I.R.' C-E-I-R (SOFTWARE) - TAB 70. 5272 River Road, Bethesda, Maryland.

- 'CALKINS 62' CALKINS, L.W. - Place of Decision Tables and DETAB-X. Proceedings Decision Tables Symposium (set 1962), 9-12, ACM, New York.
- 'CANNING 63' CANNING, R.G. (EDITOR) - Time to Consider Decision Structure Tables and EDP Design Sessions. EDP Analyzer 1: No. 4:1-10 (maio 1963).
- 'CANNING 66' CANNING, R.G.ED. - How to Use Decision Tables. EDP Analyzer 4: No. 5: 1-14 (maio 1966).
- 'CANNING 72' CANNING, R.G. ED. - That Maintenance Iceberg. EDP Analyzer 10,10 (out 1972).
- 'CANTRELL et al 61' CANTRELL, H.N. KING, J. KING, F.E.H. - Logic-Structure Tables. Comm. ACM 4,6 (jun 1961), 272-275.
- 'CANTRELL 62' CANTRELL, H.N. - Commercial and Engineering Applications of Decision Tables. Proceedings Decision Table Symposium (set 1962), 55-61.
- 'CARRICK 70' CARRICK, D.A. - Interpretation of Limited Entry Decision Table Format. (Letter to the Editor), Comput. J. 13: 220-221 (maio 1970).
- 'CAVOURAS 74' CAVOURAS, JOHN. C. - On the Conversion of Programs to Decision Tables: Method and Objectives. Comm. ACM, 17,8 (agos 1974), 456-462.
- 'CENSUS 64a' CENSUS, U.S. BUREAU OF THE - Seminar on Decision Tables: Two Years of Experience at the Bureau of the Census (CENTAB). set 30, 1964.
- 'CENSUS 64b' CENSUS, U.S. BUREAU OF THE - CENTAB: A Decision Table Preprocessor. U.S. Bur. of Census, Dept. of Commerce, Washington, D.C., dez 1964.



































































