UNIVERSIDADE ESTADUAL DE CAMPINAS

Faculdade de Engenharia Elétrica

Departamento de Semicondutores ,Instrumentos e Fotônica

RECONHECIMENTO DE TOPOLOGIAS FUNCIONAIS EM CIRCUITOS ELETRÔNICOS

Por: Sergio Aparecido Braga da Cruz

Orientador: Prof. Dr. Furio Damiani

Este exemplar corresponde à redação final da tese defendida por Sergio Aparecido Braga da Cruz e aprovada pela Comissão Julgadora em 15/04/91.

Orientador

Dissertação submetida como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica.

Campinas, abril de 1991

Dedico este trabalho

aos meus pais

Aristoteles e Eny

e a meus irmãos

Claudia e Fabio

AGRADECIMENTOS

Agradeço ao Prof. Dr. Furio Damiani pela orientação e confiança no transcorrer do desenvolvimento deste trabalho.

A CAPES pelo apoio financeiro.

As amigas Marcélia e Hsieh pelos laços de amizade criados e que tornaram este trecho da "estrada" mais fácil e agradável de ser percorrido.

Aos companheiros de república Roberto Higa, Raimes e Eli pela convivência agradável aqui em Campinas.

Aos colegas do DSIF pelo ambiente de trabalho proporcionado.

Aos meus pais pela dedicação , confiança e apoio nesta etapa da minha vida.

A todos aqueles que direta ou indiretamente tornaram possível a realização deste trabalho.

RESUMO

Este trabalho apresenta aspectos relativos à implementação de um sistema de aquisição automática de dados dedicado à criação de regras para reconhecimento de topologias funcionais em descrições de circuitos eletrônicos.

Inicialmente são estudados os paradigmas e técnicas utilizadas no desenvolvimento de sistemas de aprendizagem por exemplos.

Por fim , são apresentados um protótipo onde foram utilizadas algumas das técnicas estudadas, e os resultados obtidos.

ÍNDICE

CAPÍTULO 1

1 - INTRODUÇÃO

Os circuitos eletrônicos são representados por meio de diagramas gráficos conhecidos como esquemáticos. Os esquemáticos fornecem informações à respeito das características dos componentes eletrônicos e de como eles estão interconectados para a formação de um circuito eletrônico. A definição destas características e interconexões é feita com base nas especificações de projeto do circuito eletrônico. O desenvolvimento de um circuito eletrônico é um procedimento orientado por meta, ou seja, as especificações de projeto definem o circuito eletrônico. A meta final a que se destina o desenvolvimento do circuito é efetuada por meio de uma composição de submetas, que são realizadas por meio de uma composição de subcircuitos eletrônicos. As submetas correspondentes aos subcircuitos por sua vez, podem ser realizadas por meio de uma composição de submetas de menor nível, e assim em diante, até que as submetas correspondam a unidade funcional mínima formadora do circuito eletrônico, ou seja, os componentes eletrônicos.

Uma análise do projeto e funcionamento de um circuito eletrônico pode ser realizada com base na identificação dos subcircuitos e de suas funções dentro do circuito eletrônico global. Esta identificação pode ser clara para o autor do projeto do circuito eletrônico em situações quando as dimensões do circuito não são excessivas através da observação do esquemático do circuito. A identificação destes subcircuitos pode não ser imediata para outras pessoas que não estão envolvidas no projeto, apesar de haver uma certa regularidade nas estruturas dos circuitos eletrônicos, pois deve ser realizada através de uma inspeção visual no esquemático do circuito. Um mesmo circuito eletrônico pode ser representado em um esquemático de várias maneiras diferentes e isto dificulta a sua identificação.

O trabalho desenvolvido visa realizar a identificação

dos circuitos eletrônicos através da observação da representação destes circuitos conhecida como *netlist*, a qual pode ser criada através da observação do esquemático do circuito. O *netlist* de um circuito eletrônico é uma lista onde cada componente eletrônico do circuito é apresentado em termos de suas especificações e dos nós do circuito aos quais está conectado.

O processo de identificação dos circuitos eletrônicos é dividido em duas partes principais. Uma primeira parte está relacionada a identificação em si, a qual corresponde a um processo de reconhecimento de padrões, onde os circuitos eletrônicos são observados como os padrões a serem reconhecidos. A segunda parte está relacionada a criação de critérios que permitam a caracterização e identificação dos padrões e corresponde a um processo de aquisição automática de informação ou aprendizagem.

1.1 - Reconhecimento de Padrões

Um conjunto pode ser representado através de duas maneiras distintas. A primeira seria através da enumeração de seus elementos contituintes. A segunda maneira seria através do enunciado da lei de formação dos elementos pertencentes ao conjunto [27].

Nem sempre é possivel a representação simultânea através das duas maneiras acima.

Na segunda maneira de representar os elementos pertencentes ao conjunto, as características essenciais mínimas que distinguem os elementos que pertencem ao conjunto dos que não pertencem são abstraídas e enumeradas, definindo um elemento típico do conjunto [2]. A esta abstração poderemos chamar *Padrão*. O *Padrão* corresponderia a um exemplo perfeito. *Reconhecimento de Padrão* significa a identificação do ideal, a partir do qual um objeto foi derivado [23].

A formação do conceito de ideal, ou padrão, pode ser feita através de processos dedutivos ou indutivos. No primeiro supõe-se que o conceito de ideal é conhecido a priori pelo observador; no segundo, que o conceito de ideal é abstraído através da observação de exemplos. Este segundo processo é também chamado *aprendizagem com professor*, se os exemplos observados são rotulados como representando um ou mais elementos ideais, ou *aprendizagem sem professor*, se nenhum rótulo é fornecido [27].

Esta divisão no processo de aprendizagem corresponde a duas abordagens psicológicas sobre este tópico. Uma corresponderia à visão nativista de Chomsky, que acreditava na existência de uma grande quantidade de conhecimento inato nos recém nascidos que guiaria o processo de aprendizagem da linguagem; a outra corresponderia a visão empirista que acreditava que todo o processo de aprendizagem estaria baseado num procedimento simples e geral, o qual organizaria e otimizaria o armazenamento de informações provenientes dos orgãos de percepção [36].

Os processos mentais realizados durante a aprendizagem são ainda pouco entendidos e vários modelos foram criados na tentativa de explicá-los [36]. O advento dos computadores digitais fez surgir uma nova questão com respeito aos processos mentais: Seria possível a implementação de tais processos nestas máquinas ?

Para tratar com esses problemas surgiu o ramo da Ciência da Computação denominado *Inteligência Artificial* [2,23]. Dentro deste problema de implementação de processos mentais, a área de Reconhecimento de Padrões trata de interesses práticos, visando a automação de trabalhos tediosos que requerem o reconhecimento de padrões [23].

O problema de reconhecimento de padrões tem sido abordado de duas maneiras distintas [7,23]. Numa primeira abordagem os objetos a serem reconhecidos são representados por uma série de medidas globais, as quais tentam caracterizá-los. Estes valores podem ser interpretados como coordenadas de pontos em um espaço vetorial. Se medidas adequadas

foram tomadas, pode acontecer que pontos correspondentes a objetos derivados de um mesmo padrão estejam próximos em termos de distância geométrica. Este método é conhecido como abordagem *decisivo-teórica* [7] ou *abordagem geométrica* [27].

Esta linha de abordagem foi inicalmente seguida; porém, algumas dificuldades determinaram o aparecimento de uma nova abordagem. Apesar da sedimentação de técnicas e algoritmos conseguida durante aproximadamente duas décadas de pesquisa, não há maneira formal e geral de se determinar quais são as medidas corretas a serem tomadas para caracterizar um padrão [23].

A idéia observada para o desenvolvimento de uma nova abordagem era que os padrões complexos podiam ser descritos recursivamente em termos de padrões mais simples [27]. Esta descrição recursiva de padrões, à semelhança de frases de um texto que são descritas por meio de palavras, fez com que nesta abordagem pudesse ser aproveitada a metodologia de linguagens formais já disponíveis. Daí vem a designação desta abordagem, conhecida como *reconhecimento sintático de padrões* [7,27], ou *reconhecimento linguístico de padrões* [27]. A informação estrutural descrevendo um padrão é importante, e descreve aspectos importantes que desabilitam a classificação de um padrão dentro de uma classe inadequada [7].

Dentro do conjunto de objetos onde a informação estrutural é de grande importância encontram-se os esquemáticos de circuitos eletro-eletrônicos. Quase todos os circuitos projetados tem as características de uma linguagem, e carregam informações das intenções de um emissor mapeadas em sua estrutura. Além de representar o circuito físico, o esquemático de um circuito também funciona como uma linguagem para engenheiros elétricos [28]. Os circuitos eletrônicos apresentam um número relativamente pequeno de elementos primitivos, que são os componentes eletrônicos, os quais correspondem a elementos terminais da linguagem de descrição de circuitos eletrônicos. A composição destas primitivas permite a formação de circuitos eletrônicos por vezes complexos. Para identificação do circuito eletrônico e de seus subcircuitos componentes, o circuito é

analisado por meio de uma gramática. Esta gramática está expressa sobre componentes eletrônicos, e define uma linguagem sobre composições destes componentes, ou seja, os circuitos eletrônicos. A gramática é constituida por um conjunto de regras que descrevem como os componentes eletrônicos estão ligados para formação de um determinado circuito eletrônico. A identificação do circuito eletrônico corresponde a uma análise sintática do circuito, sendo o resultado desta análise, a identificação do circuito, e de seus subcircuitos, onde os componentes são agrupados em blocos funcionais. No trabalho desenvolvido o reconhecimento dos circuitos eletrônicos é realizado somente com base nas informações topológicas obtidas do *netlist*, ou seja, as especificações dos componentes eletrônicos são ignoradas.

1.2 - APRENDIZAGEM

A grande diversidade de circuitos eletrônicos dificulta a construção manual das regras gramaticais para identificação dos circuitos eletrônicos. Uma possibilidade para contornarmos esta dificuldade consiste na geração automática das regras gramaticais. Estas regras gramaticais ou regras para reconhecimento de circuitos podem ser geradas a partir da observação de circuitos eletrônicos exemplos através da utilização de princípios relacionados a aprendizagem automática a partir de exemplos. Através destes princípios, o circuito exemplo é generalizado de maneira criteriosa, para obtenção de uma descrição geral do circuito eletrônico que corresponde a regra de reconhecimento.

1.2.1 - Aprendizagem Humana

O processo de aprendizagem nos seres humanos é um processo que ainda não é bem conhecido, e na tentativa de explicá-lo várias teorias têm sido criadas. O resultado obtido da

aprendizagem é o *conhecimento*. Conhecimento pode ser entendido como um conjunto de fatos codificados e informações adicionais quanto à utilização destes fatos e a interação dos mesmos com a realidade [20,27]. A utilização do conhecimento nem sempre é consciente, e o processo de como este conhecimento atua na execução de uma dada tarefa nem sempre pode ser extraído da pessoa que a executa. Neste caso dizemos que existe uma *habilidade*.

O processo de aprendizagem pode ser qualificado de duas maneiras quanto ao objeto que é aprendido:(a) quanto a aquisição de conhecimento; (b) quanto ao refinamento de habilidades. Porém, a distinção entre habilidade e conhecimento é algo imperceptível, e questões filosóficas sobre o que é conhecimento e como funcionam os processos pelo qual ele é adquirido, ainda não estão resolvidas [27].

Em [20] temos uma distinção entre conhecimento e habilidade. Conhecimento seria algo consciente e a sua aprendizagem estaria associada a criação de modelos mentais e de uma linguagem pela qual poderia ser traduzido. Habilidade corresponderia a uma capacidade inata na execução de tarefas de maneira inconsciente. A aprendizagem neste caso corresponderia ao aperfeiçoamento destas habilidades por meio de repetições na execução da tarefa a ser aprendida, e uma realimentação com base na qualidade de sua execução.

No processo de aprendizagem humana, porém, não existe uma distinção clara e objetiva do que deve ser aprendido, e a classificação da aprendizagem será dada de acordo com a ênfase que é dada ao processo [20].

Várias teorias sobre aprendizagem já foram apresentadas, e como um paralelo a elas, alguns paradigmas em IA foram criados. Uma das primeiras idéias explicando a aprendizagem seria o princípio do sistema de idéia dominante ou teoria de Herbart. Neste princípio supõe-se que somente uma idéia pode estar no nível de consciência. Duas idéias não permanecem juntas no nível de consciência a não ser que possam ser unificadas, no nível consciente, transformando-se numa idéia mais geral. Uma

idéia principal faria convergir sobre o nível de consciência idéias associadas, e afastaria idéias não associadas [27]. A eficiência na aprendizagem é conseguida quando as informações a serem aprendidas estão associadas a uma idéia principal já conhecida; a aprendizagem seria então uma integração das duas idéias no nível consciente. Um paradigma de IA associado a este princípio seria a idéia de *Frames*, no qual um evento pode ser estereotipado e uma expectativa quanto as entidades participantes de um evento semelhante podem ser enumeradas [33].

A falta de formalismo e provas experimentais convincentes, fez com que surgissem outros princípios com maior aceitação. O *comportamentismo* tomou seu lugar, adotando maior rigor formal e experimental. A aprendizagem tornou-se sinônimo de condicionamento, ou seja, a criação de cadeias de estímulo-resposta, através do aperfeiçoamento da teoria psicológica do *associacionismo*[1]. Um paradigma de IA associado a este princípio seria a teoria conexionista (redes neurais) [20,27]. A busca por explicações para processos mentais a nível de conceitos, mecanismos de linguagem, e pensamento fez ressurgir o princípio de integração de idéias relacionadas para a formação de uma idéia global.

É atribuido a Piaget um estudo no qual observou-se que o processo de aprendizagem não é linear no tempo. A aprendizagem seria um processo contínuo, formado por dois estados: a assimilação e a acomodação. Na assimilação temos que toda a informação é capturada e aprendida como uma extensão de idéias já aprendidas. Na acomodação, existe uma revisão e reorganização de idéias, visando uma melhor eficiência na sua utilização. O resultado da aprendizagem não é aparente nesta fase [27]. Sob o ponto de vista de implementação de sistemas de aprendizagem, uma analogia a este princípio pode ser observada nos métodos de

---

1-associacionismo - teoria psicológica supondo que a uma sensação esta associada uma idéia, a qual leva a outra idéia, e assim por diante, formando uma cadeia de idéias derivadas.

aprendizagem intensiva de conhecimento, os quais assumem que um conhecimento próximo ao que deve ser aprendido deve estar disponível no sistema antes da aprendizagem daquele conhecimento. A aprendizagem é vista como um processo incremental de integração do conhecimento.

No desenvolvimento de sistemas artificiais, notamos que várias técnicas usadas estão apoiadas em teorias psicológicas. Com o advento dos computadores várias questões foram levantadas com respeito a sua utilização como sujeito da aprendizagem e usuário do conhecimento; porém, o objeto dessa ação, o conhecimento, ainda levanta muitas questões, independente de sua característica humana ou artificial.

1.2.2 - Aprendizagem artificial

As características do processo de aprendizagem humanas são usadas como referência na idealização de modelos e avaliação de sistemas de aprendizagem. Em analogia ao processo de criação de modelos [27] no qual supõe-se serem criados e manipulados durante o processo de aprendizagem humanas, os sistemas artificiais criam e manipulam estruturas de dados internas. Quanto mais poderosas na captura de detalhes, mais complexas se tornam estas estruturas e o seu tratamento pelo procedimento de aprendizagem [12] . No processo de aprendizagem baseado na criação de modelos é necessário que seja definida uma linguagem para descrição dos modelos e um conjunto de transformações para manipulá-los, sendo que os modelos correspondem aos padrões a serem reconhecidos. As estruturas dos padrões podem ser descritas através de gramáticas ou através de grafos relacionais [7]. As descrições estruturais dos padrões também podem ser expressas através da lógica de predicados [3,22]. Cada subcomponente ou primitiva do padrão é descrito globalmente usando-se predicados unários ou variáveis; as relações entre componentes são expressas através de funções e predicados [20].

O processo de aprendizagem corresponde a manipulações

destas estruturas, buscando uma descrição mais geral.

Na aplicação destes princípios na criação de regras de reconhecimento de circuitos eletrônicos, o circuito eletrônico apresentado ao sistema corresponde a um modelo inicial, e o procedimento para criação de regras de reconhecimento manipula este modelo com a finalidade de aumentar o seu nível de generalidade.

1.3 - Sistema desenvolvido

O desenvolvimento do sistema foi motivado pela necessidade de um visão hierárquica do circuito eletrônico, facilitando a análise do seu projeto e funcionamento. O processo de reconhecimento dos circuitos eletrônicos é tratado de maneira automatizada elevando o nível a partir do qual o circuito deve ser visualizado, ou seja, de uma visão gráfica do esquemático do circuito para uma visão funcional do circuito.

O entendimento do funcionamento dos circuitos eletrônicos é facilitado se forem reconhecidas as subpartes funcionais padrão que o compõem. O entendimento do circuito completo pode ser então determinado a partir do entendimento de suas subcomponentes, e de como elas estão relacionadas para a formação do circuito. O reconhecimento de topologias funcionais em circuitos eletrônicos constitui um auxílio a análise do funcionamento dos circuitos, destacando subcomponentes topológicas funcionais padrão, e explicitando as suas interconexões. Além disso, através do reconhecimento topológico a estrutura de um circuito eletrônico pode ser inspecionada de maneira automática, para verificação de falhas de projeto à nível topológico. Através do sistema desenvolvido, circuitos podem ser hierarquicamente apresentados, à semelhança da análise sintática de frases que destaca as categorias e subcategorias formadoras de uma frase. O trabalho desenvolvido visa a criação de uma base de dados contendo regras para reconhecimento de topologias funcionais de circuitos eletrônicos. Para a criação desta base de

dados são utilizados princípios de Inteligência Artificial relacionados com aprendizagem a partir de exemplos.

No capítulo 2 são apresentadas algumas possibilidades para a descrição dos circuitos eletrônicos dentro das abordagens utilizadas na área de reconhecimento de padrões, onde temos a descrição de padrões usando-se gramáticas ou abordagem sintática, além de linguagens derivadas do cálculo de predicados. Posteriormente algumas técnicas de manipulação de descrições são apresentadas, e constituem os mecanismos pelos quais as regras de reconhecimento dos circuitos eletrônicos podem ser geradas.

No capítulo 3 são descritos alguns aspectos importantes relacionados à implementação do sistema.

No capítulo 4 são apresentados os resultados relativos ao sistema implementado.

No capítulo 5 são apresentadas conclusões e sugestões.

CAPÍTULO 2

2.1 - DESCRIÇÃO DE PADRÕES

O processo de reconhecimento de um circuito eletrônico pode ser visto como um procedimento de reconhecimento sintático de padrões. Nesta abordagem uma classe de circuitos eletrônicos é caracterizada por meio de um conjunto de regras gramaticais que definem quais são as conexões permitidas entre os componentes eletrônico para aquela classe de circuitos. Esta visão em termos sintáticos do circuito eletrônico possibilita a utilização da metodologia de linguagens formais para a especificação da gramática para descrição de circuitos. Nesta metodologia temos disponíveis gramáticas unidimensionais e bidimensionais. A capacidade das gramáticas unidimensionais restringe a sua utilização ao tratamento de padrões simples, onde a relação entre os padrões primitivos que o compoe é a concatenação. As gramáticas bidimensionais superam esta restrição na relação entre padrões primitivos e permitem o tratamento de padrões mais complexos. Tanto as gramáticas unidimensionais quanto as gramáticas bidimensionais descrevem os padrões com base nas suas características sintáticas. A utilização de gramáticas unidimensionais é inviável para o tratamento de circuitos eletrônicos. As gramáticas bidimensionais são adequadas ao tratamento sintático dos circuitos eletrônicos, porém, a sua utilização implica num abandono de informações semânticas que possam ser utilizadas para identificação de circuitos eletrônicos.

Uma possibilidade mais geral que possibilita a descrição dos circuitos eletrônicos quanto as suas características sintáticas e semânticas são as gramáticas lógicas

Na primeira parte deste capítulo são apresentados uma introdução a linguagens formais e algumas gramáticas bidimensionais e lógicas. As gramáticas bidimensionais apresentadas são as gramáticas de árvore, gramáticas *WEB* e gramáticas *PLEX*. Como gramáticas lógicas são aprensentados o *GCD*

(gramática de cláusulas definidas) e o *APC* (cálculo de predicado anotado).

2.1.1 - ABORDAGEM SINTÁTICA

Observando-se a figura 2.1 percebe-se uma grande semelhança entre a descrição hierárquica de padrões e a estrutura de uma sentença [7]. Os nós não terminais na descrição hierárquica de padrões representando objetos ou partes de objetos dentro de uma cena correspondem a categorias e subcategorias das palavras na sentença. Os nós terminais na descrição hierárquica de padrões representando padrões primitivos correspondem às palavras na sentença.

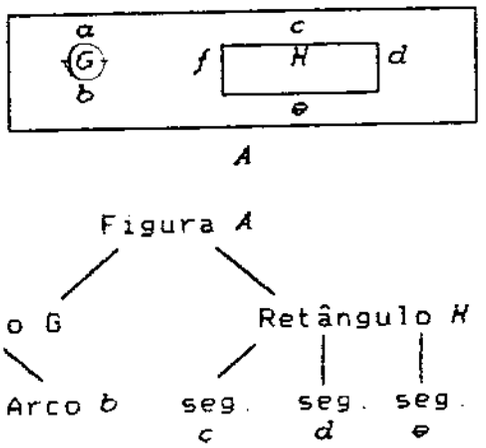


Figura 2.1 - Desenho A, e a sua descrição estrutural hierárquica.

Esta semelhança sugere que a mesma metodologia usada na descrição da estrutura de uma frase possa ser usada na descrição hierárquica de padrões. Daí a designação desta abordagem como sendo abordagem sintática. Como na estrutura de frases e sentenças, onde as sequências permitidas de palavras são determinadas por regras gramaticais, as composições permitidas

entre primitivas padrões para formação de padrões mais complexos é representada por meio de uma gramática de padrões. A linguagem definida por esta gramática é chamada de linguagem de descrição de padrões. O processo de reconhecimento de padrões passa então a corresponder a uma análise sintática do padrão, após a identificação de suas primitivas. Se o padrão for reconhecido, a análise retornará uma descrição estrutural (geralmente na forma de árvore) mostrando a construção dos padrões hierarquicamente, até o nível das primitivas padrões.

Uma outra alternativa para representação da estrutura de padrões são os grafos relacionais [7]. Nesta alternativa as relaçoes necessárias para descrição de padrões por meio de subpadrões devem ser definidas. Esta metodologia permite expressar com maior riqueza a estrutura de um padrão.

Apesar de seu maior poder de expressividade, tornando possível definir com maior precisão os detalhes que diferenciam uma classe de padrões de outra, na representação por meio de grafos relacionais não é possível se incorporar a teoria de linguagens formais para solução do problema de representação compacta e análise de padrões.

2.1.1.1 - Linguagens formais

Chomsky adaptou o sistema de regras de produção para descrição formal de gramáticas. Ao mesmo tempo que Chomsky desenvolvia as gramáticas formais para o Inglês, John Backus desenvolveu uma notação similar para definir linguagens de programação. A principal diferença entre as duas formas é que a notação na forma de Backus está limitada à gramáticas livres de contexto, enquanto Chomsky definiu, além desta, gramáticas sensíveis ao contexto e gramáticas de reescrita geral [27].

Uma gramática possui duas categorias principais de símbolos:

- os símbolos terminais, que na linguagem natural seriam representados por palavras, tais como *homem*, *casa*, *cachorro*,

etc...

- os símbolos não terminais, que na linguagem natural seriam representadas por categorias gramaticais, tais como *substantivos*, *adjetivos*, *sentença*, etc...

As regras de produção especificam como os símbolos não terminais são transformados em sentenças de linguagem (modo gerativo) ou vice-versa (modo analítico). Um símbolo não terminal especial, chamado símbolo inicial, corresponde a categoria de mais alto nível que a gramática é capaz de reconhecer, e em linguagem natural corresponderia à *sentença*. A geração de sentenças ocorre através da substituição dos símbolos não terminais (começando pelo símbolo inicial) por símbolos terminais de acordo com as regras de produção, até que se derive uma sequência de símbolos contendo somente símbolos terminais. As gramáticas desta forma são chamadas gramáticas de estrutura de frase, porque elas determinam a estrutura de uma sentença como uma hierárquia de categorias [27].

Para classificação de gramáticas, segundo Chomsky, são usadas as seguintes definições e notações.

Definição - Uma gramática de estrutura de frase G é uma quádrupla

$$G = (V_N, V_T, P, S)$$

onde:

V_N é o vocabulário de não terminais

V_T é o vocabulário de terminais

P é um conjunto de regras de reescrita ou produção denotada por $\alpha \rightarrow \beta$ onde α e β são cadeias de símbolos sobre V e com α envolvendo no mínimo um símbolo de V_N.

$S \in V_N$ é o símbolo inicial para análise de uma sentença.

A união de V_N e V_T constitue o vocabulário total V de G, e $V_N \cap V_T = \emptyset$.

Notação usada:

Σ^* é o conjunto de todas as cadeias de símbolos de comprimento finito, sobre um conjunto finito de símbolos Σ,

incluindo λ (cadeia de comprimento 0). $\Sigma^+ = \Sigma^* - (\lambda)$.

x^n é x escrito n vezes, se x for uma cadeia.

$|x|$ é o número de símbolos na cadeia x.

$\eta \overset{}{\underset{G}{\Rightarrow}} \gamma$ a cadeia η gera diretamente ou deriva outra cadeia γ se $\eta = \omega_1 \alpha \omega_2$, $\gamma = \omega_1 \beta \omega_2$ e $\alpha \rightarrow \beta$ é membro de P.

$\eta \overset{*}{\underset{G}{\Rightarrow}} \gamma$ a cadeia η gera ou deriva outra cadeia γ se existe uma sequência de cadeias $\zeta_1, \zeta_2, \ldots, \zeta_n$, tal que $\eta = \zeta_1, \ldots \gamma = \zeta_n$, $\zeta_i \overset{}{\underset{G}{\Rightarrow}} \zeta_{i+1}, i=1,2,\ldots,n-1$.

A sequência de cadeias $\zeta_1, \zeta_2, \ldots, \zeta_n$ é chamada de uma derivação γ de η.

A linguagem gerada pela gramática G é

$$L(G) = \{ x \mid x \in V_T^* \text{ tal que } S \overset{*}{\underset{G}{\Rightarrow}} x \}$$

Se G é uma gramática de estrutura de frases, então $L(G)$ é chamada de linguagem de estrutura de frase.

Uma cadeia de terminais e não terminais γ é chamada de *forma sentencial* se $S \overset{*}{\underset{G}{\Rightarrow}} \gamma$.

Uma gramática G é dita ser ambígua se existe uma sequência x de símbolos pertencentes a $L(G)$ tal que existe mais que uma sequência de derivação.

Os quatro tipos de gramáticas divididas por Chomsky são descritas a seguir [7,27]:

• Gramáticas do tipo 3 (gramáticas regulares ou de estado-finito)

As regras de produção deste tipo de gramática são da forma

$A \rightarrow aB$ ou $A \rightarrow b$

Onde $A, B \in V_N$ e $a, b \in V_T$. A, B, a e b são símbolos únicos. As linguagens geradas por este tipo de gramática são chamadas de linguagens de estado-finito ou linguagens regulares.

• Gramáticas do tipo 2 (gramáticas livres de contexto)

As produções deste tipo de gramática são da forma

$A \rightarrow B$

Onde $A \in V_N$ e $B \in V^+$.

As linguagens geradas por este tipo de gramática são

chamadas linguagens do tipo 2 ou linguagens livre de contexto.

- Gramáticas do tipo 1 ou sensíveis ao contexto

  As produções são da forma

  $\zeta_1 A \zeta_2 \rightarrow \zeta_1 B \zeta_2$

  Onde $A \in V_N$, $\zeta_1, \zeta_2, B \in V^*$, e $B \neq \lambda$.

ζ_1 e ζ_2 definem o contexto sobre o qual a regra pode ser aplicada. Estas gramáticas geram linguagens do tipo 1 ou linguagens sensíveis ao contexto.

- Gramáticas do tipo 0 ou gramáticas de reescrita geral.

Não existe restrição quanto as suas produções. O problema de decidir se uma sequência de símbolos é ou não aceita por uma gramática do tipo 0 é em geral indecidível. As linguagens geradas por este tipo de gramática são chamadas linguagens do tipo 0 .

Estas gramáticas são capazes de tratar com cadeias de símbolos, ou seja, uma sequência de dados. A capacidade descritiva destas gramáticas esta restrita à relação de concatenação entre os elementos primitivos. Esta limitação dificulta a aplicação destes tipos de gramáticas no tratamento de padrões complexos, onde a relação de concatenação não é suficiente para descrever o padrão. Para descrever padrões bidimensionais e tridimensionais são usados formalismos mais gerais. O próximo nível de generalização de gramáticas de cadeias é a gramática de árvores. Esta gramática, porém, faz parte de uma categoria conhecida como gramática de grafos. Esta categoria de gramáticas está dividida em duas subcategorias que são *"gramáticas de grafos genuínas"* e *"gramáticas de cadeias que geram grafos"* [7,23]. Nas gramáticas de grafos genuínas as regras gramaticais estão definidas sobre grafos e operam diretamente sobre eles, e nas gramáticas de cadeias que geram grafos as cadeias são representações de grafos e a gramática opera sobre estas cadeias. A gramática expressa a regra de composição dos

grafos, as cadeias mapeiam-se aos grafos.

2.1.1.2 - Gramáticas de árvore [7,11]

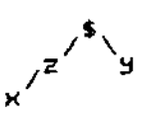
Uma árvore é um conjunto finito de um ou mais nós de maneira que

(i) Existe um único nó especial, chamado *raiz*, e

(ii) os nós restantes estão agrupados em m conjuntos disjuntos $T_1,\ldots,T_m$, cada um dos quais é chamado de uma subárvore de T.

A cada nó da árvore está associado um rótulo, que identifica o nó. Entre nós vizinhos existe um conjunto de ponteiros que descreve esta relação. A descrição de um padrão é feita relacionando os nós às primitivas do padrão e os ponteiros às possíveis relações válidas que existem entre as primitivas do padrão. Sobre os nós da árvore esta definida uma relação r que indica o número de possíveis dependentes associados a cada nó.

Ex:



$r(\$) = (2)$
$r(z) = (1)$
$r(x) = r(y) = (0)$

Uma gramática de árvore é definida por uma quádrupla:

$G_t = (V, r, P, S)$

$V = N \cup \Sigma$ onde N corresponde aos não terminais e Σ aos terminais.

(V, r) é um alfabeto "rankeado"

P é um conjunto de regras de produção na forma $T_i \rightarrow T_j$, onde T_i e T_j são árvores.

$S \in T_V$, é um conjunto de árvores iniciais. T_V denota o conjunto de árvores com nós rotulados por elementos em V.

Como para cadeias de símbolos, existe uma definição para derivação sobre árvores:

$$T_I \overset{\alpha}{\Rightarrow}_t T_{II}$$

que é lido como "T_{II} é derivado de T_I no nó α em G_t" se:

Existe uma produção $T_i \rightarrow T_j$ tal que T_i é uma subárvore de T_I sobre o nó α e T_{II} é obtido de T_I substituindo T_i por T_j sobre o nó α.

$$T_I \overset{*}{\Rightarrow}_t T_{II}$$

que é lido como "T_{II} é derivado de T_I em G_t"

Se existe uma sequência de árvores $T_I \in S$ com um nó α, T_1 com nó α_1, T_2 com nó α_2, ..., T_n com nó α_n, então.

$$T_I \overset{\alpha}{\Rightarrow}_t T_1 \ , \ T_1 \overset{\alpha_1}{\Rightarrow}_t T_2 \ , \ \ldots \ , \ T_n \overset{\alpha_2}{\Rightarrow}_t T_{II}$$

A linguagem gerada por uma gramática de árvores é um conjunto de árvores que possui somente nós terminais.

$$L(G_t) = \{ T \mid T \in T_\Sigma \text{ e } T_i \overset{*}{\Rightarrow}_t T \text{ para algum } T_i \in S \}$$

T_Σ é o conjunto de todas as árvores que possuem os nós em Σ (terminais).

Ex:

$G = ((S,A),(\$,z,x,y),S,P)$

P: $S \rightarrow \$AA$ $\quad r(\$) = (2)$

$A \rightarrow zA$ $\quad r(z) = (1)$

$A \rightarrow x$ $\quad r(x) = r(y) = (0)$

$A \rightarrow y$

Um exemplo de derivação usando esta gramática é:

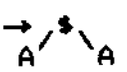
$S \Rightarrow \$AA \Rightarrow \$zAA \Rightarrow \$zxA \Rightarrow \$zxy$ $\qquad \$zxy \in L(G)$.

Gramáticas de árvore expansiva

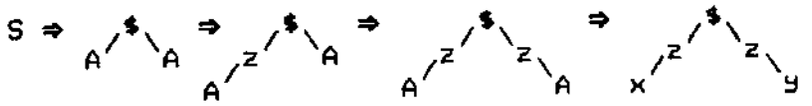
A forma expansiva de uma gramática de árvore existe quando todas as suas produções estão na forma

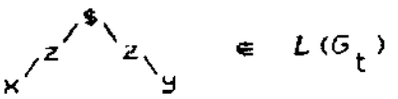
$X \rightarrow$  onde $x_1, \ldots, x_n \in N$, $x \in \Sigma$ e $n \in r(x)$.

Ex: $G_t=((S,A),r,P,S)$

P: S →  A →  A → x A → y

Uma sequência de derivação seria:





2.1.1.3 - Gramáticas WEB [7,11,27]

Grafos

Um grafo é definido por uma dupla $H=(Q,W)$, onde Q é um conjunto finito não vazio de nós, e W é um conjunto de pares de nós de Q. Cada par de W é chamado de um arco de H. Dado um arco $x = (a,b)$ dizemos que x liga a e b, e a e b são nós *vizinhos*. O ponto a e o arco x são ditos incidentes, bem como b e x. Dois arcos x e y são ditos *adjacentes* se eles são incidentes em um mesmo nó.

O grau de um nó pertencente a um grafo é o número de arcos que incidem sobre ele. Um grafo com p nós e q arcos é chamado de grafo (p,q). Dentro desta definição, um grafo não pode conter arcos que liguem nós a si mesmos (*"loops"*). Se por definição forem permitidas algumas variações sobre os grafos obteremos as seguintes estruturas.

- multigrafo - é permitido que os nós sejam conectados por meio de mais de um arco.
- pseudografo - são permitidos "*loops*" e nós conectados por mais de um arco.

Um grafo H_1 é dito ser um subgrafo de H_2 se todos os seus nós e arcos estiverem em H_2, H_2 é chamado de supergrafo de H_1. Um subgrafo estendido é um subgrafo contendo todos os nós do supergrafo correspondente.

A remoção de um nó α do grafo H resulta em um grafo $H_1 = H - \alpha$, onde $Q_1 = Q - (\alpha)$, e $W_1 = W - X$, onde X é um conjunto contendo todos os arcos que incidem em α. A remoção de um arco X de H, resulta num grafo H_1 tal que o subgrafo H_1 é um subgrafo estendido de H e $W_1 = W - X$.

Um percurso sobre um grafo H corresponde a uma sequência de nós e arcos, sendo terminado por um nó,

$$a_0, x_1, a_1, x_2, \ldots, a_{n-1}, x_n, a_n$$

tal que existe um arco x_i ligando a_{i-1} à a_i .

Se $a_0 = a_n$ diz-se que o percurso é *fechado*, caso contrário é *aberto*. Grafos fechados definem ciclos se o percurso tem um número de arcos maior que 3. Um grafo sem ciclos é chamado *acíclico*.

Durante o percurso, se a ordem dos nós no par que define um arco deve ser levada em consideração, dizemos que o grafo é dirigido.

Um padrão pode ser mapeado na estrutura dos grafos através de duas maneiras distintas. Uma delas é usar os nós relacionando-os à primitivas do padrão através dos rótulos; os arcos descrevem então as relações permitidas entre as primitivas do padrão. Outra forma é obtida invertendo-se a convenção anterior. Uma "*web*" (rede) é por definição um grafo não dirigido, com nós rotulados, ou seja a primeira convenção.

Gramáticas de Rede [7,11]

Novamente usando a analogia com gramáticas de estrutura de frases, temos o conceito de regra de substituição. Na gramática de estrutura de frases, toda a informação para a substituição está disponível na regra; porém, para a substituição em redes são necessárias informações adicionais. Estas informações especificam de que maneira uma subrede α de uma rede ω é substituida pela subrede β , ou seja, como a subrede β é encaixada em ω, em substituição a α.

Uma gramática web é definida por uma quádrupla.

$$G = (V_N, V_T, P, S)$$

onde:

V_N corresponde aos símbolos não terminais

V_T corresponde aos símbolos terminais

Estes símbolos rotulam os nós na rede.

P é o conjunto de regras de produção ou reescrita. Cada produção é definida por uma tripla (α,β,f), onde:

$f : N_\beta \times N_\alpha \rightarrow 2^V$

α e β são redes.

N_β é um conjunto de nós da rede β.

N_α é um conjunto de nós da rede α.

V é um conjunto de rótulos.

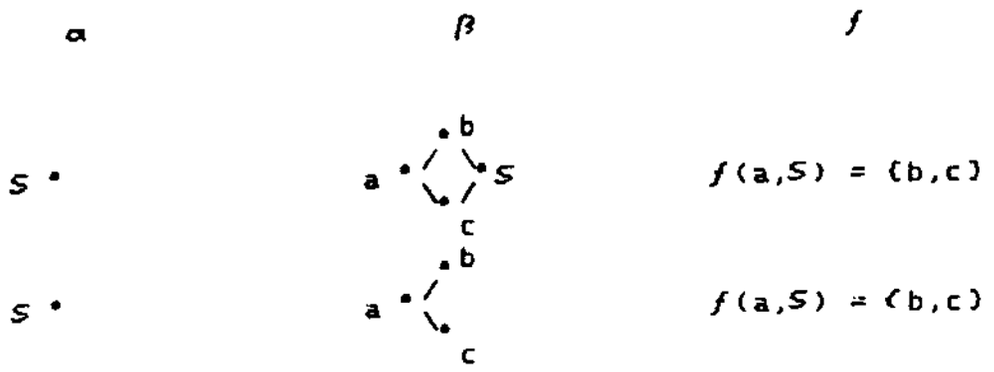
2^V é um conjunto de subconjuntos de rótulos.

f especifica o encaixe de β no lugar de α e tem a forma $f(n,m)$, onde $n \in N_\beta$ e $m \in N_\alpha$. O valor da função especifica a que vizinhos de m n deve ser conectado. O termo *normal* será usado no lugar da função quando não houver ambiguidade na substituição.

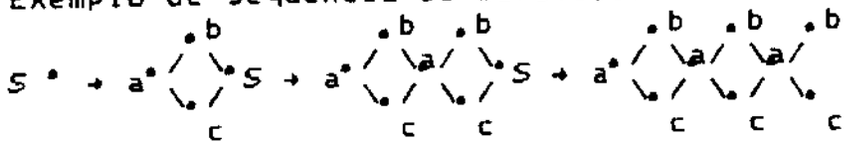
$S \subseteq V_N$ é o conjunto de redes iniciais.

$V = V_N \cup V_T$ é o vocabulário da gramática.

Ex: $G_w = (V_N, V_T, P, (S))$ onde $V_N = (S)$, $V_T = (a,b,c)$, e P é o seguinte conjunto de triplas:



Exemplo de sequência de derivação:



Deve ser observado que na primeira substituição sobre S, a informação $f(a,S) = (b,c)$ não é satisfeita; porém, não seria necessária, pois não existe ambiguidade nesta substituição.

As regras da gramática de rede definidas acima são irrestritas, isto é, nenhuma restrição é imposta às construções das regras. Esta generalidade dificulta o uso prático desta abordagem. Uma restrição às regras pode ser feita através da incorporação de um contexto de aplicabilidade das regras. Uma regra sensível ao contexto seria aquela regra (α,β,f) onde existe um símbolo não terminal A em α, de maneira que $\alpha - (A)$ é um subconjunto de β, ou seja, um contexto. A substituição efetiva obtida substituindo-se β por α em uma rede seria igual a substituição de um único nó da rede.

Para α igual a um único nó diz-se que a regra é livre de contexto.

2.1.1.4 - Gramáticas PLEX [7,11]

A palavra "PLEX" é derivada da palavra "plexus", e está relacionada com um arranjo entrelaçado de partes. Nestas

gramáticas são definidos elementos primitivos que possuem vários pontos de junção a outras primitivas. Estas primitivas são conhecidas como NAPE (*"n attaching-point entity"* ou entidades com n pontos de ligação). A interligação destas entidades básicas formam uma estrutura "PLEX".

Definição:

A gramática PLEX é uma sêxtupla.

$$G_p = (N, \Sigma, P, S, I, i_o)$$

onde:

N é um conjunto não vazio de NAPE's chamado de vocabulário de não terminais.

Σ é um conjunto de NAPE's chamado de vocabulário de terminais.

P é um conjunto finito de regras de produção ou substituição.

$S \in N$ é um NAPE especial chamado NAPE inicial.

I é um conjunto finito de símbolos chamado identificadores.

$i_o \in I$ é um identificador especial chamado identificador nulo.

$N \cap \Sigma = \emptyset$ e $I \cap (\Sigma \cup N) = \emptyset$.

Cada ponto de conexão de um NAPE é rotulado por um elemento distinto pertencente a I. As ligações entre NAPE's são feitas através de pontos de junção específicos, diferentes do identificador nulo. Um conjunto de NAPE's é dito estar conectado se existe um percurso entre quaisquer dois elementos do conjunto de NAPE's.

De acordo com o tipo das regras de substituição as gramáticas PLEX podem pertencer a um dos seguintes tipos.

- Gramáticas PLEX irrestritas.
- Gramáticas PLEX sensíveis ao contexto.
- Gramáticas PLEX livres de contexto.

Gramáticas PLEX irrestritas

Possuem regras na forma:

$$\psi\Gamma_{\psi}\Delta_{\psi} \rightarrow \omega\Gamma_{\omega}\Delta_{\omega}$$

ψ é chamado de lista de componentes do lado esquerdo da regra, Γ_{ψ} é a lista de junções do lado esquerdo e Δ_{ψ} a lista de pontos de ligação do lado esquerdo. ω, Γ_{ω} e Δ_{ω} são as listas equivalentes no lado direito da regra.

$\psi\Gamma_{\psi}$ e $\omega\Gamma_{\omega}$ definem as estruturas de NAPE's ou subplex's que participam do processo de substituição. Δ_{ψ} e Δ_{ω} ditam a maneira como a subplex $\omega\Gamma_{\omega}$ deve substituir a subplex $\psi\Gamma_{\psi}$ em alguma PLEX.

Para definição do subplex é necessário uma especificação dos NAPE's que o compõem e da maneira como eles estão interligados. Isto é feito por ψ e Γ_{ψ} para o lado esquerdo, e ω e Γ_{ω} para o lado direito da regra, tal que:

$$\psi = a_1\ a_2 \ldots a_i \ldots\ a_n$$
$$\omega = b_1\ b_2 \ldots b_j \ldots\ b_n$$

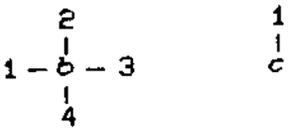
onde a_i e b_j são NAPE's simples, chamadas *componentes*. Através de ψ e ω são definidas as componentes de cada subplex e uma ordem entre elas. Os pontos de interligação entre um ou mais NAPE's formam uma *junção*. Γ_{ψ} e Γ_{ω} são listas de junções definindo como os NAPE's estão interligados. Cada junção corresponde a um campo formado por uma lista de identificadores indicando a contribuição de cada NAPE na formação da junção. A não contribuição de um NAPE, na formação de uma junção é indicada pelo identificador nulo. A correspondência entre o NAPE e o identificador com o qual ele contribui na formação da junção é obtida observando-se a ordem definida por ψ ou ω dentro da junção, o i-ésimo elemento na j-ésima junção é um identificador do i-ésimo elemento na sequência de componentes. As ligações das subplex's $\psi\Gamma_{\psi}$ e $\omega\Gamma_{\omega}$ à plex onde ocorre a substituição são

definidas por Δ_ψ e Δ_ω, que correspondem a listas de pontos de ligação. A correspondência entre os elementos de Δ_ψ e Δ_ω é definida através da ordem dos elementos na lista. Cada ponto de ligação em Δ_ω é formado por campos que indicam como cada componente contribui na formação do ponto de ligação. Δ_ψ define os mesmos pontos de ligação que Δ_ω, porém Δ_ψ os define no contexto da estrutura hospedeira e Δ_ω os define no contexto da subestrutura no lado direito que será encaixada na estrutura no lugar de $\psi\Gamma_\psi$.

Ex: regra

$$A(1,2,3) \rightarrow bc(41)(10,20,30)$$

Onde A é um NAPE não terminal e b e c são NAPE's terminais com as estruturas na forma



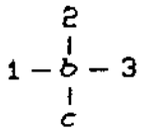
Para esta regra temos:

$\psi = A$, $\omega = bc$,

$\Gamma_\psi = \emptyset$, $\Gamma_\omega = (41)$,

$\Delta_\psi = (1,2,3)$, $\Delta_\omega = (10,20,30)$.

$\omega\Gamma_\omega$ define a subplex que irá substituir o não terminal A na PLEX. $\Gamma_\omega = (41)$ indica que esta estrutura é formada ligando-se o ponto de junção rotulado por 4 da NAPE b, com o ponto de junção rotulado por 1 da NAPE c. Isto corresponde à subestrutura



$\Delta_\psi = (1,2,3)$ tem três campos que indicam os pontos de ligação de A. $\Delta_\omega = (10,20,30)$ indica que na subestrutura o NAPE b contribui com as junções 1, 2 e 3 e que c

não contribui (indicado pelo identificador 0). O ponto de ligação 1 em Δ_ψ da plex hospedeira corresponde ao ponto de ligação 10 da subplex, o 2 ao 20, e 3 ao 30.

Para o uso das regras plex sem que surjam ambiguidades devem ser observadas as seguintes convenções:

- Um NAPE não se liga a ele mesmo.
- Todas as interconexões entre NAPE's são aquelas declaradas na lista de junções.
- Toda interconexão de todo NAPE no lado esquerdo(direito) deve, ou conectá-lo a outro NAPE no lado esquerdo(direito), ou a algum ponto de ligação.

Gramáticas PLEX sensíveis ao contexto

Possuem a regra na forma:

$$A\ \psi_1\ \Gamma_{\psi_1}\ \Gamma_{A\psi_1}\ \Delta_A \rightarrow x\ \psi_1\ \Gamma_x\ \Gamma_{\psi_1}\ \Gamma_{x\psi_1}\ \Delta_x$$

Onde A é um NAPE simples e x e ψ_1 são listas de NAPE's; a subplex definida por $\psi_1\Gamma_{\psi_1}$ corresponde ao contexto no qual a NAPE A pode ser substituida por $x\Gamma_x$.

Cada elemento da regra pode ser interpretado como:

Γ_{ψ_1} é uma lista de junções usada na descrição das interconexões entre as NAPE's da lista ψ_1.

Γ_x é uma lista de junções usada na descrição das interconexões entre as NAPE's da lista x.

$\Gamma_{A\psi_1}$ é uma lista de junções usada na descrição das interconexões entre a NAPE A e os elementos da lista ψ_1; esta lista descreve as junções entre o NAPE A e a subestrutura $\psi_1\Gamma_{\psi_1}$.

$\Gamma_{x\psi_1}$ é uma lista de junções descrevendo a interconexão entre as subestruturas $x\Gamma_x$ e $\psi_1\Gamma_{\psi_1}$.

Δ_A e Δ_x são listas de pontos de ligação que dão a correspondência entre os pontos de encaixe de A e os pontos de ligação de $x\Gamma_x$.

Gramáticas PLEX livre de contexto

A restrição imposta à gramática sensível ao contexto é determinada pela estrutura definida por $\psi_1\Gamma_{\psi_1}$(contexto). A eliminação deste termo torna as regras livres de contexto, ou seja, as gramáticas livres de contexto têm a regra na forma:

$$A\ \Delta_A \rightarrow x\ \Gamma_x\ \Delta_x$$

Exemplo: aplicação da gramática PLEX na geração de registradores de deslocamento de comprimento arbitrário.

$$G_P = (N, \Sigma, P, S, I, i_O)$$

Onde:

N = (⟨SHFT STGE⟩,⟨SHFT RGSTR⟩),

Σ = (⟨FF⟩,⟨A⟩,⟨D⟩),

S = ⟨SHFT RGSTR⟩,

I = (0,1,2,3,4,5),

i_o = 0,

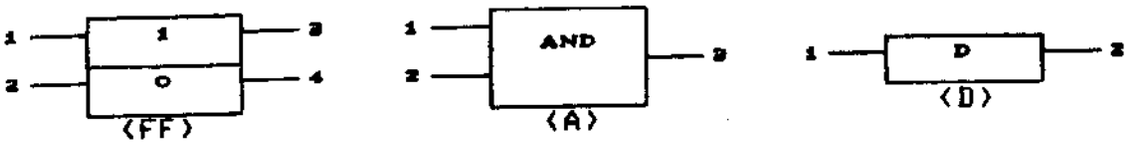
e as produções são

⟨SHFT RGSTR⟩(1,2,3,4,5) →
⟨SHFT STGE⟩⟨SHFT RGSTR⟩(41,52,33)(10,20,33,04,05)

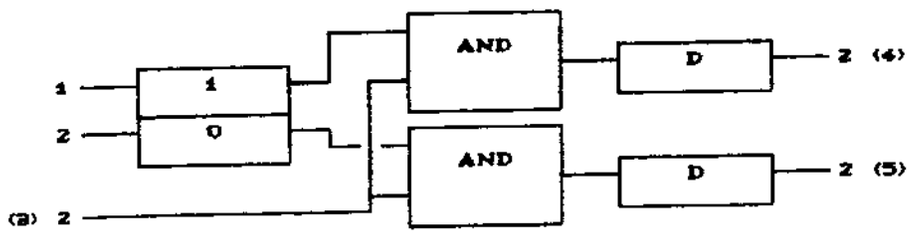
⟨SHFT RGSTR⟩(1,2,3,4,5) →
⟨SHFT STGE⟩()(1,2,3,4,5)

⟨SHFT STGE⟩(1,2,3,4,5) →
⟨FF⟩⟨A⟩⟨A⟩⟨D⟩⟨D⟩(31000,40100,02200,03010,00301)
(10000,20000,02200,00020,00002)

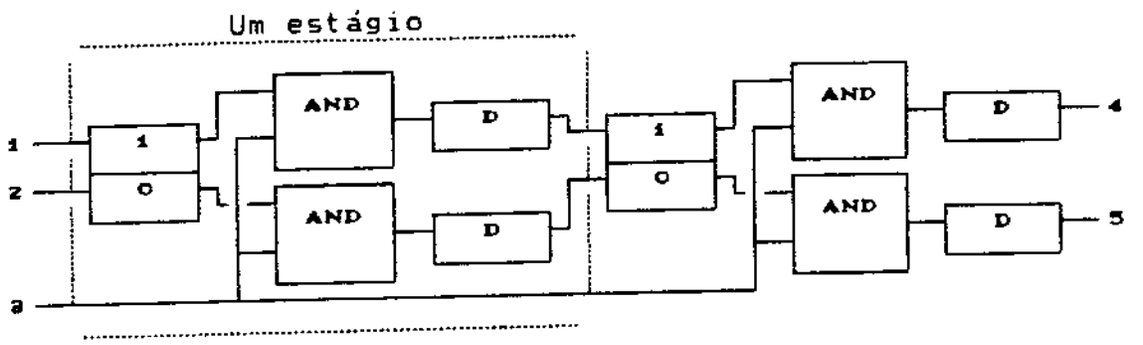
Os terminais NAPE's nesta gramática correspondem a flip-flops, portas E e linhas de atraso:



A segunda regra de produção gera a seguinte estrutura correspondendo a um estágio do registrador de deslocamento:



Um exemplo de dois estágios de um registrador de deslocamento gerado pela gramática é:



2.1.2 - LÓGICA

Da Grécia antiga até o século XIX aproximadamente, a *lógica* objetivava o estudo de formas de raciocínio usando linguagem e formas de reflexões comuns. A linguagem comum, devido a sua característica ambígua, foi gradativamente substituída por formas de expressão mais precisas. A definição de uma linguagem própria e precisa permitiu a unificação de princípios gerais, e determinou uma independência contextual da lógica. Esta abstração de princípios possibilitou a aplicação da lógica como meio de abstração e validação de teorias.

Das suas origens na linguagem comum podemos informalmente especificar alguns componentes da linguagem usada na lógica. As partículas da linguagem natural *e*, *ou*, *se...então* e *não* deram origem aos conectivos lógicos. Em linguagem natural, o *e* e o *ou* correspondem a conjunções e são colocadas entre orações no intuito de se expressar fatos simultâneos e fatos alternativos, respectivamente. A expressão *se...então* exprime fatos condicionados, e a partícula *não* exprime a negação de um fato. Os conectivos lógicos constituem uma abstração destas partículas da linguagem natural. Dentro da linguagem comum a realidade é expressa nomeando-se entidades, suas características e interrelações. Os termos e predicados em lógica visam este mesmo objetivo. Os termos são expressões da linguagem lógica que direta ou indiretamente nomeiam entidades na descrição de fatos. Os predicados são expressões que nomeiam as características das entidades, e suas interrelações [4,27].

Com descrições de fatos da realidade expressos por meio dos conectivos lógicos, termos e predicados correspondendo a uma abstração da realidade, a lógica busca a determinação da consistência destas descrições, através da utilização de métodos para verificar se uma descrição é verdadeira ou falsa, dada uma descrição da realidade. Estes métodos são conhecidos como *cálculos* [3].

O cálculo proposicional e o cálculo de predicados [3,22] são exemplos destes métodos. A diferença entre estes

métodos reside no grau de refinamento com que podem traduzir os fatos da realidade. A partir da linguagem usada no cálculo de predicados foram desenvolvidas linguagens e morfismos derivados. Dentre eles temos a gramática de cláusulas definidas (GCD) que por sua vez foi derivada da unificação do formalismo das gramáticas livres de contexto (GLC) e do subconjunto da lógica conhecido como cláusulas definidas [24]. O cálculo de predicado anotado (CPA) é um exemplo de linguagem derivada do cálculo de predicados visando uma maior facilidade nas aplicações em aprendizagem [20].

2.1.2.1 - CÁLCULO DE PREDICADO ANOTADO [20]

A sintaxe do APC (*"annotated predicate calculus"* ou cálculo de predicado anotado) aproxima-se da linguagem natural no sentido em que usa conectivos na ligação de termos, como na linguagem natural partículas ligam entidades. A semântica do APC esta associada a semântica do cálculo de predicado. O significado de uma sentença em APC será o significado da sentença equivalente a ela no cálculo de predicados. A equivalência é feita por meio de regras de tradução de uma linguagem para outra. Com o APC a descrição de fatos se torna mais compacta e a manipulação de expressões durante processos de aprendizagem se torna mais simples. A principal diferença entre o cálculo de predicados e o APC esta na presença dos conectivos $\wedge$ (conjunção ou conectivo *e*) e $\vee$ (disjunção ou conectivo *ou*) entre termos. Estes conectivos são nomeados operadores internos. Os símbolos destes operadores são idênticos aos símbolos dos operadores conectando predicados, os quais são nomeados operadores externos. A determinação de um operador interno ou externo é feita observando-se o contexto onde ele é aplicado, ou entre termos do APC ou entre predicados do APC, respectivamente.

Termos em APC

Os termos em APC são classificados em dois tipos: o termo elementar ou *etermo* e o termo composto ou *ctermo*. O etermo corresponde ao termo do cálculo de predicados e o ctermo é uma composição de etermos ou um termo elementar cujos argumentos são ctermos. A composição de etermos é realizada por meio da disjunção interna (símbolo "$\vee$") e da conjunção interna (símbolo "$\wedge$"). Abaixo estão alguns exemplos de ctermos

$$vermelho \wedge azul$$
$$altura(caixa1 \wedge caixa2)$$

vermelho, *azul*, *caixa1* e *caixa2* são constantes. A transformação de ctermos em etermos é realizada por meio das seguintes regras de transformação. É assumido que f é uma função n-ária(possui n argumentos), onde $n-1$ argumentos são representados por A, e $t1$ e $t2$ são etermos.

$$f(t1 \vee t2, A) \leftrightarrow f(t1, A) \vee f(t2, A) \qquad (1)$$
$$f(t1 \wedge t2, A) \leftrightarrow f(t1, A) \wedge f(t2, A) \qquad (2)$$

Através destas transformações , o segundo exemplo do ctermo é traduzido para a forma:

$$altura(caixa1) \wedge altura(caixa2)$$

As disjunções internas devem ter prioridade no processo de transformação. É importante observar que os ctermos não são avaliáveis como sendo verdadeiros ou falsos, como podem insinuar os operadores internos. Os ctermos, como os termos em cálculo de predicado, são usados como argumentos de predicados, e os operadores internos terão influência somente na avaliação destes predicados.

Predicados em APC

São divididos em dois grupos, sendo eles o predicado elementar e o predicado composto. O predicado elementar possui a mesma sintaxe e semântica dos predicados do cálculo de predicados. O predicado composto é aquele que possui como argumento no mínimo um ctermo. A semântica de um predicado composto é aquela obtida pelo seu predicado equivalente no cálculo de predicados, o qual é obtido pelas regras de equivalência abaixo:

$$P(t1 \vee t2, A) \models P(t1, A) \vee P(t2, A) \qquad (3)$$

$$P(t1 \wedge t2, A) \models P(t1, A) \wedge P(t2, A) \qquad (4)$$

onde P é um predicado n-ário, $t1$ e $t2$ são etermos, e os últimos $n-1$ argumentos de P são representados por A. Nestas regras de equivalência é possível verificar como os operadores internos são transformados em operadores externos e de que maneira atuam na avaliação de um predicado. Os argumentos de P formados por disjunção de etermos devem ter prioridade durante a aplicação da regra. Os argumentos de P que são ctermos, devem inicialmente ser transformados em uma composição de etermos usando-se as regras (1) e (2). A seguir vê-se um predicado composto do APC e a sua forma expandida em predicados elementares.

- $foram(maria \wedge mãe(joão), cinema \vee teatro)$
- $foram(maria, cinema) \wedge foram(mãe(joão), cinema) \vee foram(maria, teatro) \wedge foram(mãe(joão), teatro)$

Os predicados compostos em APC apresentam uma grande semelhança com textos em linguagem natural, e expressam de maneira explícita e compacta os fatos, propiciando uma maior legibilidade.

Textos relacionais

Um texto relacional em APC possui a forma:

Termo1 rel Termo2

onde *Termo1* e *Termo2* são termos de APC e *rel* representa uma das seguintes relações: =, ≥, >, ≤, <. Esta forma equivale ao predicado *rel(Termo1,Termo2)*. Esta expressão declara que as entidades resultantes da avaliação de *Termo1* e *Termo2* pertencem a relação *rel*. A avaliação do texto relacional é aquela resultante do seu predicado elementar equivalente obtido por meio das regras de transformação (1), (2), (3), e (4).

Temos um seletor referencial quando, na expressão *Termo1 rel Termo2*, *Termo1* é um termo elementar não constante e *Termo2* é uma constante ou uma disjunção interna de constantes pertencentes ao domínio do *Termo1*. Neste caso *Termo1* é chamado referido e *Termo2* é chamado de referência. A seguir vê-se um exemplo de texto relacional.

$$cor(p1 \vee p2) = vermelho \vee azul$$

Durante a expansão deste texto relacional são seguidos os seguintes passos:

1 - $cor(p1 \vee p2) = vermelho \vee azul$

2 - $=(cor(p1 \vee p2), vermelho \vee azul)$

3 - $=(cor(p1), vemelho \vee azul) \vee =(cor(p2), vermelho \vee azul)$

expressão obtida de 2 aplicando-se (1)

4 - $=(cor(p1), vermelho) \vee =(cor(p1), azul) \vee$
$=(cor(p2), vermelho) \vee =(cor(p2), azul)$

expressão obtida de 3 aplicando-se (1)

O texto no terceiro passo é formado por uma disjunção de seletores referênciais.

O símbolo .. é usado para simplificar a expressão de um seletor caso a referência do seletor referencial seja formada por

um intervalo ordenado e consecutivo de valores disjuntos do domínio de *Termo1*, e *Termo1* seja um descritor linear (pagina 53). Por exemplo:

$$tamanho(p) = 2 \vee 3 \vee 4$$

pode ser reescrito como

$$tamanho(p) = 2 \,..\, 4$$

Um predicado arbitrário $P(t1, t2, \ldots, tn)$ pode ser escrito na forma de um seletor referencial como:

$$P(t1, t2, \ldots, tn) = verdadeiro$$

As expressões em APC utilizam como texto básico o seletor referencial, e expressões mais complexas em APC são obtidas conectando-se seletores referênciais por meio de conectivos lógicos ($\sim$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$) e quantificadores. Em analogia aos quantificadores do cálculo de predicados, o APC possui o quantificador numérico. Sua forma é:

$$\exists_{(I)\ v}\ ,S[v]$$

onde I é um conjunto indexado, formado por um conjunto de inteiros e $S[v]$ é uma expressão em APC. O conjunto I é um conjunto de inteiros definindo o número de valores de v para os quais $S[v]$ é verdadeiro. Como exemplo temos:

$$\exists_{(2..8)\ v},S[v]$$

Esta expressão declara que existem de 2 a 8 valores de v que tornam $S[v]$ verdadeiro.

Os quantificadores $\exists$ e $\forall$ do cálculo de predicados podem ser escritos de maneira equivalente usando-se os quantificadores numéricos $\exists_{(I)}$ como é descrito a seguir.

$\exists\ v\ S[v]$ é equivalente a $\exists_{(\geq 1)}\ v\ S[v]$

e

$\forall\ v\ S[v]$ é equivalente a $\exists_{(K)}\ v\ S[v]$

onde K é o número de possíveis valores de v. Esta segunda equivalência pode ser expressa também como:

$$\exists_{v1,v2,\ldots,vk}\ ,S[v1,v2,\ldots,vk]$$

onde $v1,v2,\ldots,vk$ são valores distintos da variável v para os quais $S[v1,v2,\ldots,vk]$ é verdadeiro.

Um exemplo de expressão complexa em APC é:

$$\exists_{p0,p1,p2,p3}([contem(p0,p1,p2,p3)][emcima(p1\wedge p2,p3)]$$
$$[comprimento(p1) = 3..5][peso(p1) > peso(p2)]$$
$$[cor(p1) = vermelho \vee azul][forma(p1\wedge p2\wedge p3) = cúbica])$$

Esta expressão pode ser traduzida como:

" Um objeto $p0$ contém somente as partes $p1$, $p2$ e $p3$. As partes $p1$ e $p2$ estão em cima da parte $p3$, $p1$ tem um comprimento entre 3 e 5, o peso de $p1$ é maior que o de $p2$, a cor de $p1$ é vermelha ou azul e a forma das três partes é cúbica."

2.1.2.2 - Gramática de Cláusulas Definidas (GCD)[24]

Uma definição precisa para uma linguagem é realizada por meio de um conjunto de regras conhecido como gramática. Dentre os tipos de gramáticas classificados por Chomsky, está a gramática livre de contexto (GLC). O GCD surgiu como resultado da unificação dos formalismos do GLC com a representação em cálculo de predicados. Esta idéia foi concebida por Colmerauer e Kowalski que imaginaram expressar regras da gramática livre de contexto como textos de uma forma mais restrita do cálculo de predicados conhecido como cláusulas de Horn ou cláusulas definidas [24]. Usando-se este novo formalismo, o processo de

análise da cadeia de símbolos de uma linguagem transforma-se numa prova lógica da cadeia, tendo como axiomas um conjunto de cláusulas definidas as quais constituem a nova descrição da linguagem.

Esta unificação do formalismo do GLC com a representação de cláusulas definidas resultou em uma extensão natural para a GLC. A GCD possui como características herdadas do GLC a modularidade e clareza na descrição de linguagens; além disso, existe um conjunto de resultados estabelecidos sobre GLC's que são úteis no projeto de algoritmos de análise. O GCD extende as capacidades do GLC de três maneiras distintas:

- É possível por meio de GCD expressar a dependência contextual entre elementos de uma sentença.
- O uso de GCD permite que sejam construídas estruturas em árvore de formas arbitrárias, de maneira independente das estruturas recursivas da gramática.
- O GCD permite que condições extras em número ilimitado sejam incluídas nas regras gramaticais, tornando a análise dependente de cálculos auxiliares.

Cláusulas definidas

O conjunto de cláusulas definidas [3] constitui um subconjunto da lógica, o qual serviu de base para a linguagem de programação PROLOG (*"Logic Programming"*)[3]. A sintaxe das cláusulas definidas é derivada do cálculo de predicados e a sua semântica pode informalmente ser atribuída segundo duas maneiras distintas, a semântica procedural e a semântica declarativa. Na semântica procedural as cláusulas definidas são interpretadas como instruções de *como* deve ser realizado determinado procedimento. Na semântica declarativa as cláusulas definidas são interpretadas como textos declarativos definindo *qual* é a solução para um determinado problema.

Sintaxe de cláusulas definidas

A sintaxe das cláusulas definidas é formada por dois elementos principais, o termo e a cláusula. O termo é derivado diretamente do cálculo de predicados e a cláusula equivale a um predicado do cálculo de predicados.

Termo

É usado para definir entidades abstraídas e que serão expressas pela linguagem. O termo é uma *constante*, uma *variável* ou um *termo composto*. Seguindo a mesma notação da linguagem PROLOG, temos que as constantes podem ser divididas em dois grupos : os inteiros e os átomos.

Os átomos são formados por uma sequência de caracteres, podendo opcionalmente estar entre apóstrofes, caso haja possibilidade de confusão na notação. Os inteiros representam quantidades inteiras, e além deles novos tipos de constantes numéricas também podem estar definidas, tais como os números reais. As variáveis devem ser imaginadas como objetos particulares, porém indefinidos. Elas são representadas simbolicamente por meio de uma sequência de caracteres, sendo o primeiro um caracter alfabético maiúsculo. Os termos compostos são formados por um funtor (funtor principal do termo) e uma sequência de um ou mais termos chamados argumentos. O funtor, por sua vez, é composto pelo seu nome e aridade, que é o número de argumentos do termo composto. Exemplos dos tipos de termos são:

- inteiros: 0 1 999
- átomos: := = 'Prolog' [] const1
- variáveis: X Y Volume
- termo composto: *ponto(X,Y,Z)*

  onde o funtor é *ponto/3*, *ponto* é o nome do funtor e 3 é a aridade.

Dentro do conjunto de termos compostos existe uma classe importante conhecida como *listas*. O funtor das listas é

igual a ./2 . A aridade 2 expressa que existem dois argumentos, sendo que o primeiro representa o primeiro elemento da lista, ou cabeça da lista, e o segundo argumento representa o resto ou cauda da lista. Uma lista vazia é representada pelo símbolo []. No PROLOG uma lista pode ser representada de maneira equivalente enumerando-se os seus elementos entre colchetes. A explicitação dos argumentos cabeça e cauda nesta notação equivalente é realizada usando-se o símbolo |. Exemplos de listas são:

.(1,.(2,.(3,[]))) ou [1,2,3]

[X|L]

[a,b|L]

A variável X representa a cabeça de uma lista e a variável L representa a cauda.

Cláusulas

Um programa em lógica consiste de uma sequência de cláusulas. Cada cláusula possui a seguinte estrutura:

$$P :- M_1, \ldots, M_n .$$

onde P é denominada a cabeça da cláusula e é formada no máximo por uma meta. Os elementos de M_1 à M_n formam uma sequência de submetas e definem o corpo da cláusula. Os operadores "," na estrutura da cláusula podem ser interpretados como a conjunção lógica (operador $\wedge$). Para efeito de clareza algumas vezes é usado o operador ";" que é interpretado como disjunção lógica. Os operadores tem prioridades definidas de maneira que o operador ":-" tem precedência sobre o operador ";" e este tem precedência sobre o operador ",". O operador ; sempre pode ser eliminado da cláusula, substituindo a cláusula que o contém por duas outras cláusulas equivalentes. As variáveis que ocorrem em uma cláusula, têm seu escopo limitado a elas.

A meta de um programa em lógica equivale a uma chamada de procedimento nas linguagens de programação convencionais. Sua

forma é essencialmente a mesma de um termo, mas distingui-se dele pelo contexto no qual aparece no programa. O funtor principal de uma meta é chamado de um predicado. A estrutura da cláusula acima pode ser interpretada declarativamente ou proceduralmente como segue:

- Interpretação declarativa

" P será verdadeiro se M_1 for verdadeiro, M_2 for verdadeiro ,..., e M_n for verdadeiro ".

- Interpretação procedural

" Para satisfazer a meta P, satisfaça as submetas M_1, $M_2,\ldots,M_n$ ".

Duas outras formas de cláusula podem ser derivadas da estrutura básica acima. Se o corpo da cláusula for vazio teremos a *cláusula unitária* que terá a forma:

$$P.$$

onde P é a cabeça da cláusula. Esta cláusula é interpretada como "P é verdadeiro" e "A meta P está satisfeita", usando-se respectivamente a interpretação declarativa e procedural. Se a cabeça da cláusula for vazia teremos uma *questão* que terá a forma:

$$?- M_1,\ldots,M_n.$$

onde $M_1,\ldots,M_n$ correspondem ao conjunto de submetas da cláusula. O texto "$M_1, M_2,\ldots,$ e M_n são verdadeiros ?" é a interpretação declarativa da questão. "Satisfaça as submetas $M_1, M_2,\ldots,M_n$" corresponde a interpretação procedural da questão.

Um conjunto em sequência de cláusulas com cabeças formadas por um mesmo funtor principal definem um procedimento para um predicado particular. A seguir vê-se um exemplo de cláusula

```
avô(X,Z):- (mãe(X,Y);pai(X,Y)),pai(Y,Z).
```

Esta expressão pode ser interpretada como:

"Para quaisquer X,Y,Z, o avô de X é Z se a mãe ou o pai de X é Y, e Z é pai de Y".

A cláusula acima pode ser expressa pela forma equivalente com a eliminação do operador ";" como é mostrado a seguir :

```
avô(X,Z):- mãe(X,Y),pai(Y,Z).
avô(X,Z):- pai(X,Y),pai(Y,Z).
```

O conjunto de cláusulas acima constitui um procedimento, pois as cláusulas possuem o mesmo funtor *avô/3* relativo ao predicado avô.

Semântica Procedural ou Declarativa

Para definirmos semântica procedural é necessário que tenhamos em mente o significado de *instância* de um termo ou cláusula e o conhecimento do processo de *unificação* [3,22].

Os termos são interpretados como entidades do universo. As variáveis, como termos simples ou como argumentos em termos compostos, especificam de maneira relativa, uma entidade do universo. Cada substituição desta variável por uma entidade específica do universo resulta em uma instância do termo que contém a variável. O processo de unificação encontra a instância comum mais geral entre dois termos.

A instância de uma cláusula consiste na extensão das substituições aplicadas à cabeça da cláusula, ao corpo da cláusula, uma vez que as variáveis na mesma cláusula devem denotar uma mesma entidade. Na semântica declarativa as cláusulas definindo um programa em lógica são interpretadas como descrevendo quando as metas podem ser consideradas verdadeiras. Sua definição informal é:

"Uma meta é verdadeira se ela é a cabeça de uma cláusula instância, e se cada uma das submetas no corpo desta cláusula (se existirem) também forem verdadeiras".

Nesta semântica, a ordem das submetas no corpo da cláusula, e das cláusulas formando o programa lógico, não são levadas em consideração.

Na semântica procedural as cláusulas de um programa em lógica são interpretadas como procedimentos a serem executados a partir da chamada de procedimento que corresponde a meta. Sua definição informal é:

"Para executar uma meta o sistema deve buscar pela primeira cláusula cuja cabeça unifica com a meta. Se existir tal cláusula, a cláusula é dita ativada e a execução deve prosseguir sobre cada uma das submetas no corpo da cláusula (se existir) seguindo da esquerda para a direita.

Caso não seja encontrado uma cláusula que unifique com a meta, o sistema deve realizar o procedimento de *'backtracking'*. Neste procedimento a cláusula mais recentemente ativada deve ser rejeitada, e todas as substituições necessárias à sua unificação com a meta devem ser desfeitas. Em seguida deve ser reconsiderada a meta original, que ativou a meta rejeitada, buscando-se um cláusula subsequente que possa ser ativada ".

Esta interpretação de um programa em lógica é a base para a implementação prática do interpretador PROLOG. Seguindo esta interpretação vemos que a execução de uma meta pode ter seu tempo alterado de acordo com a ordem das cláusulas e das submetas no corpo de uma cláusula.

Temos abaixo um exemplo de procedimento para concatenar duas listas.

```
concatenar([X|L1],L2,[X|L3]):- concatenar(L1,L2,L3).
concatenar([],L,L).
```

Neste procedimento *concatenar(L1,L2,L3)* significa que a lista *L1* concatenada com a lista *L2* resulta em *L3*.

Pela semântica declarativa temos que a meta *concatenar([a],[b],[a,b])* é verdadeira pois ela corresponde a uma instância da primeira cláusula de *concatenar/3*, cuja submeta é verdadeira. O corpo da cláusula instanciada corresponde a meta

concatenar([],[b],[b]), sendo verdadeira pois corresponde a cabeça da cláusula unitária de *concatenar/3*. Sob a semântica procedural a meta é vista como uma chamada de procedimento e o resultado da execução deste procedimento será uma instância verdadeira da meta. Considerando a meta,

?- concatenar(X,Y,[a,b]).

temos os seguintes passos durante a execução:

1) A meta *concatenar(X,Y,[a,b])* unifica com a cabeça da primeira cláusula *concatenar([X|L1],L2,[X|L3])*. As variáveis após a instanciação terão os seguintes valores:

-para *concatenar(X,Y,[a,b])*: $X = [a|L1]$ e $Y = L2$.

-para *concatenar([X|L1],L2,[X|L3])*: $X = a$, $L2 = Y$ e $L3 = [b]$.

A primeira cláusula é ativada e a nova meta passa a ser *concatenar(L1,L2,[b])*.

2) A meta *concatenar(L1,L2,[b])* unifica com a primeira cláusula. As variáveis terão os seguintes valores:

-para *concatenar(L1,L2,[b])*: $L1 = [b|L1']$ e $L2 = L2'$.

-para *concatenar([X|L1],L2,[X|L3])*: $X = b, L1 = L1', L2 = L2'$ e $L3 = []$.

A primeira cláusula é ativada e a nova meta passa a ser *concatenar(L1',L2',[])*.

3) A meta *concatenar(L1',L2',[])* não unifica com a primeira cláusula. A cláusula unitária unifica para os seguintes valores das variáveis:

-para *concatenar(L1',L2',[])*: $L1' = []$ e $L2' = []$.

-para *concatenar([],L,L)*: $L = []$.

A meta *concatenar(L1',L2',[])* é portanto verdadeira.

4) A meta *concatenar(L1,L2,[b])* é verdadeira para $L1 = [b|L1']$ e $L2 = L2'$, ou seja $L1 = [b|[]]$ e $L2 = []$.

5) A meta *concatenar(X,Y,[a,b])* é verdadeira para $X = [a|L1]$ e $Y = L2$, ou seja, $X = [a|[b|[]]]$ e $Y = []$. X equivale a lista $[a,b]$.

A primeira resposta para a questão é X = [a,b] e Y = [].

Se esta resposta for rejeitada, uma nova solução será procurada através do mecanismo de *"backtracking"*.

6) A cláusula correspondente a última escolha é rejeitada e a meta retoma o seu valor original. O último ponto de escolha corresponde ao passo 3. A meta original é então *concatenar(L1',L2',[])*. Como não há uma regra subsequente, ocorre novo *"backtracking"*.

7) Um novo *"backtracking"* ocorre no ponto de escolha correspondente ao passo 2. A meta original neste ponto é *concatenar(L1,L2,[b])*. A primeira cláusula é rejeitada. A cláusula unitária é escolhida e unifica com a meta para os seguinte valores de variáveis:
-para *concatenar(L1,L2,[b])*: L1 = [] e L2 = [b].
-para *concatenar([],L,L)*: L = [b].

A meta *concatenar(L1,L2,[b])* é verdadeira para L1 = [] e L2 = [b].

8) A meta *concatenar(X,Y,[a,b])* é verdadeira para X = [a|L1] e Y = L2, ou X = [a|[]] e Y = [b].

A segunda resposta é X = [a] e Y = [b].

Para a meta *concatenar([1],[2],L)* temos a seguinte execução.
1) A meta *concatenar([1],[2],L)* unifica com a primeira cláusula com os seguintes valores das variáveis.
-para *concatenar([1],[2],L)*: L = [1|L3].
-para *concatenar([X|L1],L2,[X|L3])*: X = 1, L1 = [] e L2 = [2].

A clausula é ativada e a nova meta passa a ser *concatenar([],[2],L3)*.

2) A meta *concatenar([],[2],L3)* não unifica com a primeira, porém unifica com a cláusula unitária para os seguintes valores das variáveis:
-para *concatenar([],[2],L3)*: *L3 = [2]*.
-para *concatenar([],L,L)*: *L = [2]*.

A meta *concatenar([],[2],L3)* é verdadeira para *L3 = [2]*.

3) A meta *concatenar([1],[2],L)* é verdadeira para *L = [1|L3]*, ou seja, *L = [1|[2]]*.

Uma resposta para a questão é *L = [1,2]*.

Observando estes dois exemplos podemos verificar algumas características importantes de programas em lógica:

- O processo de unificação constroe, passo a passo, estruturas mais complexas e seleciona elementos específicos dentro destas estruturas.
- Os argumentos não têm suas funções restritas, ou como entrada, ou como saída. Os argumentos são bidirecionais e a característica de entrada ou saída é atribuida de acordo com as necessidades de instanciação dos termos durante a execução de uma meta.
- Através do mecanismo de "*backtracking*", um procedimento pode retornar respostas alternativas.
- O resultado da execução de uma meta é obtido gradativamente, de acordo com a chamada de suas submetas componentes. Neste processo observamos que a cada meta ou submeta específica, as variáveis são o meio de transferência das respostas que são desconhecidas, e que serão determinadas por procedimentos posteriores. Cada meta fornece uma resposta incompleta em termos de variáveis.
- O programa em lógica e os seus dados possuem formas semelhantes.

GCD

Usando-se o formalismo de cláusulas definidas, o GLC pode ser extendido dando origem ao GCD. A forma obtida para o GCD é a seguinte.

(1) Os símbolos não-terminais de GLC são substituidos por termos simples e compostos em GCD.

(2) A regra de GCD, além de terminais e não-terminais, possui uma sequência de chamadas de procedimento escritos entre chaves. Estes procedimentos secundários definem condições extras a serem satisfeitas pela regra.

Exemplo de uma regra em GCD:

substantivo(S) → [W],{raiz(W,S),é_substantivo(S)}

onde *[W]* é um terminal.

substantivo(S) é um não-terminal.

raiz(W,S) e *é_substantivo(S)* são procedimentos auxiliares.

A interpretação para a regra é:

" Uma frase é um substantivo se ela consiste somente da palavra *W*, onde *W* possui a forma raiz *S*, e *S* é um substantivo".

O GCD herda características importantes derivadas das cláusulas definidas:

- Argumentos auxiliares podem ser acrescentados às regras, com a finalidade de construção de estruturas em árvore associadas ao processo de análise de determinada frase. As estruturas em árvore são formadas através do processo de unificação, e possuem formas independentes das estruturas recursivas da gramática.
- Os procedimentos secundários restringem o número de resultados possíveis para uma determinada análise.
- Argumentos comuns podem ser acrescidos entre não-terminais no corpo da regra, servindo como meio de transferência de informações contextuais.

2.2 - APRENDIZAGEM

O processo de identificação dos circuitos eletrônicos consiste de um processo de reconhecimento sintático de padrões. Nesta abordagem uma classe de circuitos eletrônicos é caracteriza por meio de um conjunto de regras sintáticas que descrevem como os vários componentes eletrônicos podem ser interconectados para formarem um circuito da classe.

Nos casos onde os padrões a serem reconhecidos são simples e em pequeno número, a criação destas gramáticas pode ser realizada diretamente por um especialista através de inspeção e análise de amostras dos padrões [11]. Este porém não é o caso dos circuitos eletrônicos. Além de formarem composições complexas, o número de circuitos eletrônicos é ilimitado. Por este motivo, para criação do conjunto de regras sintáticas, fez-se uso de princípios de aquisição automática de informação ou aprendizagem. Através do uso destes princípios o conjunto de regras sintáticas é gradualmente incrementado pelo acréscimo de novas regras sintáticas que são obtidas pela observação de exemplos de circuitos eletrônicos de uma determinada classe.

Para a descrição dos circuitos eletrônicos especificando um exemplo para uma determinada classe de circuitos eletrônicos são utilizadas informações topológicas obtidas das suas *netlist*'s. Estas descrições topológicas são o ponto de partida para a criação de novas regras sintáticas.

Os sistemas de aprendizagem automática encontrados na literatura são formados por duas partes principais: a) representação da informação a ser aprendida; b) mecanismos de manipulação da representação. Neste sentido a informação topológica descrevendo um circuito eletrônico constitui a representação. Os mecanismos de manipulação da descrição do circuito eletrônico visam generalizar de maneira criteriosa esta informação e desta maneira obter uma descrição geral do circuito. Esta descrição geral corresponde a regra sintática ou regra de reconhecimento do circuito.

Nesta segunda parte do capítulo 2 são apresentados os

princípios básicos dos sistemas de aprendizagem automática.

Supondo-se a existência de um determinado conhecimento K que explique um conjunto de dados observados I, podemos imaginar a existência de uma função B que mapeia o conhecimento K neste comportamento observável I. Um algoritmo de aprendizagem L seria aquele que produziria, a partir do comportamento observado I, o conhecimento K. Supondo que B seja conhecida e inversível, o algoritmo de aquisição de conhecimento ideal L seria a sua inversa, ou seja, $L(I)=B^{-1}(I)=K$. O relacionamento entre B e L é determinado por uma teoria de indução [12]. O processo de aquisição de conhecimento a partir dos dados é conhecido como aprendizagem.

Alguns trabalhos encontrados na literatura, que consistem na implementação de modelos para o processo de aprendizagem, são apresentados abaixo:

*Aprendizagem de estruturas a partir de exemplos(Winston) [33,34]

Os dados observados são descritos através de modelos em forma de redes semânticas. O procedimento de aprendizagem refina o modelo inicial para um determinado conceito a ser aprendido através da comparação deste com amostras fornecidas criteriosamente por um instrutor do sistema. O instrutor tem a função de ordenar as amostras de maneira adequada e de rotular a amostra como sendo um exemplo ou um contra-exemplo do conceito. Com estas informações o procedimento promove a generalização e especialização de relações e objetos na rede semântica representativa do conceito, de maneira que esta represente a descrição de um elemento típico do conceito, porém não descreva os seus contra-exemplos.

*Sistema THOTH-P (VERE)[29-32]

Aprende a definição de operadores através da observação da descrição das situações antes e após a sua aplicação. A representação das situações e da atuação dos operadores é feita através do sistema de produção relacional (RPS). O RPS consiste

de uma fusão da idéia de produção do tipo 0 para cadeia de caracteres com a idéia de relação entre objetos, dando como resultado a definição de produção sobre relações. As situações antes e após a aplicação de um operador são descritas por meio de uma conjunção de relações, e a modificação da situação resultante da aplicação do operador é descrita pela produção relacional.

*Sistema SPROUTER(Hayes-Roth)[12-14]

Possui como princípio de indução o casamento de semelhanças, o qual detecta as características comuns entre dois ou mais objetos descritos por meio da estrutura relacional parametrizada (PSR). Esta estrutura organiza por grupos semânticos relacionados, as relações descrevendo um determinado objeto. O resultado do procedimento de casamento de semelhanças consiste de um conjunto de relações comuns, abstraídas dos objetos analisados.

*Sistema INDUCE (Michalsky)[20,21]

Um conjunto de técnicas abstraídas de trabalhos anteriores sobre aprendizagem é aplicada sobre amostras de descrições de conceitos. O conjunto de técnicas é dividido em generalizações seletivas e generalizações contrutivas. As descrições são expressas por meio de uma extensão do cálculo de predicado de 1ª ordem. O procedimento básico generaliza as descrições de exemplos utilizando-se destas técnicas. O objetivo do procedimento é encontrar uma ou mais descrições gerais a partir dos exemplos do conceito, de modo a caracterizá-lo, sendo que nenhum contra-exemplo pode ser expresso por estas descrições.

*Sistema ID3 (Quinlan)[25,26]

O procedimento de inferência usado em ID3 constroe uma regra em forma de árvore de decisão a partir da apresentação de um conjunto composto por exemplos e contra-exemplos de um

determinado conceito. Cada elemento deste conjunto é descrito por meio dos valores de seus atributos relativos a características padrões. O sistema ID3 seleciona dentro das características padrões, aquelas que promovem a máxima discriminação entre exemplos e contra-exemplos. Novas características discriminantes são escolhidas, caso a característica inicial não seja suficiente para discriminar todos os exemplos e contra-exemplos do conjunto apresentado. Cada característica define um nó da árvore de decisão; os atributos possíveis para cada característica são os ramos a partir daquele nó.

*SNPR (Wolff) [35,36]

Consiste de um modelo psicológico para aquisição de linguagem. Tem como premissa básica a suposição de que o processo de aprendizagem consiste do refinamento e desenvolvimento de estruturas buscando a eficiência na execução de determinada tarefa. No caso da aprendizagem de uma linguagem, as estruturas iniciais obtidas de sentenças brutas são gradativamente refinadas e desenvolvidas, observando-se os princípios de compressão de dados. Estes princípios organizam as informações contidas nas sentenças apresentadas de maneira a otimizar o espaço ocupado pelo seu armazenamento.

Algumas tentativas de unificação e padronização de princípios básicos [1,20] utilizados em sistemas de aprendizagem foram feitas a partir da observação dos vários sistemas implementados. Como características comuns estes sistemas podem ser observados como uma composição de dois elementos principais: uma representação para os dados necessários à aprendizagem, e um conjunto de transformações sobre esta representação. A representação descreve os fatos observados e as tranformações são usadas na manipulação destes fatos.

A obtenção de um conjunto de hipóteses H respaldando e explicando um conjunto de fatos observados F é efetuado através de transfomações sobre estes fatos. O conhecimento básico a

respeito do domínio dos fatos observados, define e restringe estas transformações. Transformações visando obter hipóteses a partir de fatos são ditas generalizações, o inverso é a especialização. As hipóteses H obtidas podem explicar os fatos, ou por consequência lógica ($H \Rightarrow F$), ou por *implicação fraca*, onde F é uma consequência parcial de H, e nem todos os componentes do fato são explicados por H.

As restrições impostas às transformações delimitam em um número finito as hipóteses H que podem ser obtidas de F. A partir do conjunto F uma hipótese pode ser gerada, ou por generalização descritiva ou aquisição de conceitos. Na aquisição de conceitos observamos a formação de novos conceitos como uma integração de conceitos mais simples. No processo de generalização descritiva a meta é encontrar uma descrição geral abstraída de um conjunto de dados [20].

2.2.1 - Linguagens de descrição

No mundo real existe uma infinidade de níveis de detalhamento disponíveis, através dos quais um determinado acontecimento pode ser descrito. Esta riqueza de níveis de detalhamento deve ser restrita, observando-se as necessidades mínimas para tratamento de um determinado problema.

Um conjunto de fatos observados deve corresponder a uma parcela do mundo real de relevância para o processo de aprendizagem. Para descrição destes fatos observados e de suas generalizações, observamos a necessidade de uma linguagem que expresse os detalhes relevantes. O cálculo de predicados e as suas formas derivadas constituem linguagens adequadas para descrição da realidade no nível adequado à aprendizagem. Isto porque o cálculo de predicados possui conceitos sedimentados, algoritmos consolidados e validados e comercialmente existem linguagens de alta eficiência implementadas, como por exemplo o PROLOG, para o tratamento de problemas lógicos.

As variáveis, predicados e funções do cálculo de predicados são designados então de *descritores* e descreverão os fatos observados bem como as hipóteses para explicação destes fatos que são chamados *afirmações indutivas*.

Descritores

Da adequabilidade dos descritores na tradução dos fatos observados através de uma descrição, depende a eficiência do procedimento de aprendizagem. Descritores inadequados descrevem os fatos observados sobre um ponto de vista inútil para efeito de aprendizagem, ou seja, as afirmações indutivas derivadas das descrições destes fatos não corresponderão a uma generalização que seja de utilidade para classificação dos eventos desejados.

Quanto a sua capacidade em descrever os fatos observados, os descritores podem ser classificados em três tipos distintos [20]:

- *Descritores com relevância completa*

Os descritores definidos para o processo de aprendizagem expressam todos os aspectos necessários. A afirmação indutiva corresponde a uma forma mais geral desta descrição, derivada dela lógica ou matematicamente.

- *Descritores parcialmente relevantes*

Os descritores definidos para o processo de aprendizagem descrevem de maneira ambígua ou irrelevante os fatos observados; porém, algumas características são relevantes (alguns descritores são relevantes). No processo de aprendizagem é necessário então a identificação e seleção dos descritores relevantes e, a partir deste ponto, derivar a afirmação indutiva.

- *Descritores com relevência indireta*

Alguns descritores definidos inicialmente no processo de aprendizagem que não são diretamente relevantes, podem ser integrados para definirem um novo descritor que é relevante. No processo de aprendizagem é então necessário identificar descritores não diretamente relevantes, a partir dos quais é possível a criação de um descritor relevante. As afirmações indutivas são obtidas por generalizações baseadas nos descritores derivados.

Os descritores descrevem características e relações entre objetos no domínio do problema. Este conjunto de objetos define o domínio do descritor. O conjunto de objetos para os quais os descritores são válidos é um fator que limita o nível de generalização sobre estes descritores. A estrutura do domínio do descritor define as técnicas de generalização aplicáveis a ele.

Quanto ao tipo de domínio, podemos classificar os descritores em três tipos [20]:

- *Descritores nominais*

O domínio destes descritores são nomes e símbolos representando os objetos. Os nomes e símbolos não obedecem nenhuma ordem ou estrutura.

Ex: cor_cabelo(pessoa,castanho)

- *Descritores lineares*

O domínio dos descritores é um conjunto ordenado, ou indiretamente os descritores definem funções que mapeiam-se em conjuntos ordenados.

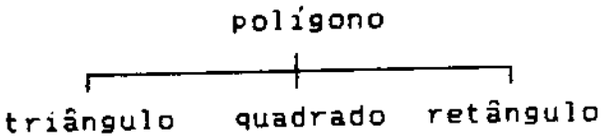
Ex: temperatura(temp)

distância(Cidade1,Cidade2,Dist)

- *Descritores estruturados*

O domínio possui uma estrutura em árvore relacionando os seus elementos de maneira hierárquica de generalização. Os elementos mais próximos da raiz são os mais gerais.

Ex:



Os descritores também podem ser organizados de maneira hierárquica de generalização; isto definirá as operações de generalização aplicáveis sobre eles.

2.2.2 - Regras de generalização

O processo de obtenção de uma afirmativa indutiva a partir de fatos observados corresponde a um processo de busca de solução em um espaço de estados [22], onde os estados correspondem a descrições simbólicas, e o estado inicial corresponde a descrição dos fatos observados. As operações sobre estados para obtenção de novos estados correspondem as regras de generalização, especialização e reformulação (regras de inferência) aplicáveis à uma descrição. O estado meta é uma afirmação indutiva que implica nos fatos observados e satisfaz as restrições impostas pelo conhecimento inicial do problema.

Dado um conjunto de fatos observados F, o número de hipóteses H que podem ser criadas e que implicam em F é potencialmente infinito [20]. A limitação desse processo de criação de hipóteses é observado em dois níveis. O primeiro seria

na definição de descritores que delimitam o nível de detalhamento da descrição dos fatos observados e afirmativas indutivas, ou seja, define o espaço de estados de soluções possíveis. Uma segunda restrição seria imposta às regras de transformação de um estado em outro. O conhecimento inicial sobre o problema define as transformações possíveis. Mesmo com estas restrições o número de hipóteses consistentes com os fatos observados é grande [20], e faz-se necessário uma medida *heurística*[2] delimitando o número de afirmações indutivas ao número desejado. Uma afirmação indutiva pode ser selecionada segundo vários critérios. Alguns deles são:

- As afirmações indutivas são expressões de descritores. Se o número de descritores de uma afirmação indutiva for elevado, o entendimento e avaliação destas descrições por um ser humano ficam comprometidas. Um critério a ser adotado então, pode ser o número de descritores usado na afirmação indutiva.
- Pode-se atribuir um custo para cada descritor na afirmação indutiva. As afirmações indutivas incorporando descritores de custo mínimo seriam selecionadas. Uma função pode ser criada sobre os custos dos descritores mapeando-os em um custo total da afirmação indutiva. As afirmações indutivas com mínimo custo seriam selecionadas. Este custo pode ser atribuído levando-se em consideração fatores computacionais, tais como o espaço de memória ocupado ou o tempo de avaliação da afirmação indutiva por exemplo, ou visando destacar aspectos mais interessantes na forma da afirmação indutiva.

Dadas as descrições H, F, $F1$, C e admitindo que F é uma descrição de fatos observados, podemos classificar as três transformações possíveis sobre as descrições [20] como:

2-heuristica - medida empirica do grau de adequabilidade de uma resposta.

-Especialização: $F \Rightarrow C$

Obtem a partir de F uma descrição C que é consequência lógica dos fatos observados.

-Reformulação: $F \Leftrightarrow F1$

Transforma a descrição F em uma descrição equivalente $F1$.

-Generalização: $H \Rightarrow F$

Dada a descrição F, obtem H, tal que H implica em F.

Uma característica do processo de generalização é a preservação de consistência da falsidade. Os eventos que falsificam uma descrição devem falsificar também a sua generalização.

Isto decorre da lei de contraposição; dado que $H \Rightarrow F$ temos que $\sim F \Rightarrow \sim H$ [22].

Algumas tolerâncias podem existir quanto a generalização, de maneira que os fatos observados são explicados parcialmente pelas hipóteses geradas; neste caso temos que H implica fracamente em F [20].

O processo de generalização está apoiado na integração de um exemplo positivo de determinado evento, à uma descrição pré-existente ou modelo, e tem o efeito de tornar o modelo inicial mais tolerante quanto aos aspectos observados no exemplo, de maneira que eventos semelhantes ao exemplo que sejam apresentados posteriormente serão reconhecidos pelo modelo [33].

O processo de especialização está apoiado na integração de contra-exemplos ou exemplos negativos a uma descrição ou modelo já existente, visando impor uma limitação nos modelos derivados de uma generalização (correção de supergeneralização [36],[33]). Se os contra-exemplos visam atuar somente sobre uma característica do modelo, são chamados *"near miss"* [1,20,22,29,34]; caso o contra-exemplo atue sobre várias características ao mesmo tempo então é designado *"far miss"* [1,20,32].

O procedimento de aprendizagem de exemplos pode se basear em exemplos e contra-exemplos, caso em que os processos de generalização e especialização estão ambos presentes. Em procedimentos de aprendizagem baseados somente em exemplos, somente o processo de generalização está presente.

A aprendizagem pode se dar usando como estruturas de suporte regras ou descrições. Uma regra pode ser inicialmente fornecida para o tratamento de determinado problema, e a atuação inadequada da mesma é usada como base para a formulação de regras corretas [1]. Uma regra é considerada falha se associar uma conclusão incoerente às suas premissas (falha factual), ou por não disparar adequadamente (falha de controle). A falha de controle pode ser, ou por omissão, quando a regra deveria disparar mas não disparou, ou por co-omissão, quando a regra dispara conjuntamente com uma outra regra correta. A criação de uma regra nova corresponderia a correção de uma regra inicial nula. Na aprendizagem usando-se descrições [20] o procedimento de aprendizagem esta associado principalmente a informações obtidas somente das descrições de fatos observados, através de generalização. O processo de generalização não garante que a descrição geral obtida seja útil ou plausível. A única garantia é que a descrição geral é válida para os eventos que serviram de base para a sua criação.

Se a generalização produz uma descrição contendo somente descritores provenientes da descrição do evento observado, temos uma *generalização seletiva*, caso contrário, temos uma *generalização construtiva*. A generalização seletiva não modifica o espaço de afirmações indutivas possíveis, enquanto que a generalização construtiva modifica. Esta modificação é entendida como uma alteração na representação do evento, ou seja, um redirecionamento do enfoque dado na descrição do evento, resultando numa descrição mais geral com descritores caracterizando aspectos não diretamente observados na descrição inicial do evento.

2.2.2.1 - Aprendizagem por manipulação de regras existentes

2.2.2.1.1 - Discriminação

Consiste basicamente em restringir a generalidade de uma regra, impondo condições extras. É suposto que a regra tenha funcionado corretamente em alguma situação anterior, porém no exemplo atual a regra apresenta uma falha de co-omissão.

Para a definição da condição extra a ser acrescentada na regra são necessárias três informações:

- Um conjunto de substituições [22] das variáveis da regra, quando da sua atuação incorreta. Este conjunto de substituições constitue o contexto de rejeição da regra.
- Um conjunto de substituições [22] das variáveis da regra, quando de sua atuação correta. Este conjunto de substituições constitue o contexto de seleção da regra.
- Um conjunto de descritores e de seus complementares.

O procedimento consiste de três passos:

- Realizar as substituições apresentadas no contexto de rejeição sobre o conjunto de descritores.
- Realizar as substituições apresentadas no contexto de seleção sobre o conjunto de descritores.
- Avaliar os dois conjuntos resultantes e selecionar os descritores que foram avaliados verdadeiros, derivando do contexto de seleção, e falsos, derivando do contexto de rejeição. Este conjunto de descritores constitue um conjunto de *discriminantes* que podem ser adicionados conjuntivamente à regra inicial falha.

Ex: Afirmações verdadeiras na base de dados.

```
homem(João).
viúvo(João).
pai(João,Ana).
bancário(João).
homem(Pedro).
pai(Pedro,Mário).
bancário(Pedro).
```

Regra: H ⇒ C

homem(X),pai(X,Y) ⇒ casado(X)

H ≡ homem(X),pai(X,Y)

C ≡ casado(X)

Contexto de seleção:

(X/Pedro,Y/Mário)

Contexto de rejeição:

(X/João,Y/Ana)

Descritores:bancário(X) viúvo(X) ~bancário(X) ~viúvo(X)

Avaliação:

(X/Pedro,Y/Mário)	(X/João,Y/Ana)
bancário(Pedro) = V	bancário(João) = V
viúvo(Pedro) = F	viúvo(João) = V
~bancário(Pedro) = F	~bancário(João) = F
~viúvo(Pedro) = V	~viúvo(João) = F

discriminantes encontrados: H'= {~viúvo(X)}

Nova regra: H,H' ⇒ C

homem(X),pai(X,Y),~viúvo(X) ⇒ casado(X)

Neste exemplo somente um discriminante foi encontrado. Nesta situação temos um *"near miss"*. Caso um outro discriminante fosse encontrado (por exemplo supondo-se *bancário(Pedro)* = falso), teríamos um *"far miss"*. Este segundo caso pode ser tratado criando-se uma nova regra para cada discriminante encontrado. Assim teríamos:

homem(X),pai(X,Y),~viúvo(X) ⇒ casado(X)

homem(X),pai(X,Y),~bancário(X) ⇒ casado(X)

A segunda regra constitui uma regra irrelevante, e exemplos posteriores poderão evidenciar isto. Quando ocorrer um exemplo em que esta segunda regra disparar inadequadamente e não for possível encontrar um discriminante para corrigi-la, será o momento dela ser descartada.

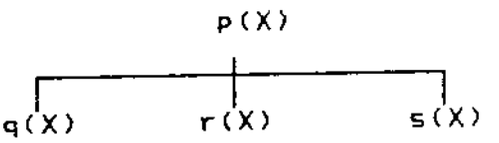
Observa-se que o conjunto de descritores tem um papel fundamental nesta abordagem. Os descritores devem ser fornecidos pelo usuário, e correspondem a conceitos primitivos que os seres

humanos podem criar com base na análise do problema. Esta análise e a criação dos descritores não podem ser simulados no computador.

2.2.2.1.2 - Focalização

Este algoritmo é baseado na manipulação de uma estrutura hierárquica de descritores, organizados na forma de árvore. Cada nó da árvore esta rotulado por um descritor. O descritor rotulando o nó raiz é sempre avaliado como verdadeiro. Os nós filhos são rotulados por descritores exaustivos e mutuamente exclusivos, de maneira que o nó pai é uma generalização dos nós filhos e equivale lógicamente somente a um deles.

Ex:



p(X) equivale somente a um dos descritores de <q(x),r(X),s(X)>.

Os descritores de uma mesma árvore são aplicados aos mesmos argumentos. Dado um exemplo de um evento somente um dos nós terminais será verdadeiro. Este exemplo é dito especificar o descritor. Um descritor e a sua negação podem ser explicitamente relacionados por uma árvore composta por um nó raiz e dois nós filhos rotulados com o descritor e a sua negação. Uma árvore somente com um nó raiz e dois nós filhos é chamada de uma árvore minimal.

Dado um conceito, uma regra que o caracteriza ou descreve pode ser aproximada incrementalmente através de um refinamento dos descritores que podem ser usados na sua descrição, usando-se para isto as árvores de descritores.

Uma regra pode ser enunciada usando-se os descritores mais gerais, os quais se encontram mais próximos da raiz da árvore, ou usando-se os descritores mais específicos, que se

encontram mais próximos dos nós terminais da árvore.

O processo de refinamento delimita os descritores plausíveis de serem incorporados a parte condicional da regra, através da fixação de dois rótulos na árvore. São eles o *marcador superior e inferior*. Dada uma árvore de hierarquia, temos que os descritores acima do marcador superior estão fora do escopo da definição do conceito, os descritores abaixo do marcador inferior estão dentro do conceito e os descritores que estão entre estes dois marcadores constituem uma região de indecisão. Um nó ou seu descritor associado é considerado estar acima de um marcador se ele não pertence a subárvore onde o marcador é o nó raiz; caso contrário é considerado estar abaixo do marcador.

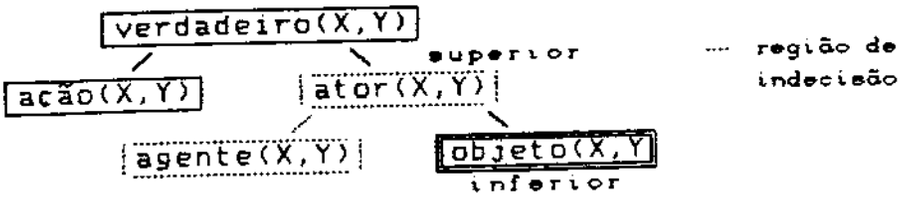


Figura 2.2 - Árvore representando trecho de regra para aquisição de linguagem, onde estão definidos os rótulos superior e inferior.

O algoritmo de focalização atua no sentido de diminuir a região de indecisão. O máximo refinamento possível ocorrerá quando o marcador superior e o inferior de uma árvore coincidirem sobre um nó. O descritor associado ao nó é então incorporado a regra de maneira definitiva. O algoritmo de focalização usa como entradas a conclusão da regra a ser corrigida e um conjunto de árvores de descritores com as respectivas posições dos marcadores superior e inferior; esta entrada define a chamada *regra bruta*. Cada *regra bruta* esta definida por um conjunto específico de árvores que a descrevem implicitamente. O algoritmo atualiza a posição dos marcadores diminuindo a região de indecisão.

A forma da regra não esta explicitamente definida, sendo sua representação parcial obtida usando-se os descritores dentro da região de indecisão. A regra pode ter uma forma mais

geral se no processo de formação da regra os descritores sobre os nós com marcadores superior forem agrupados conjuntivamente na parte de condição da regra. A regra assim formada será pouco restrita e poderá ser suscetível a erro de co-omissão. Uma forma mais específica da regra pode ser obtida, usando-se na sua parte condição os descritores sobre os nós com os marcadores inferior. A regra obtida será mais restrita e suscetível ao erro de omissão.

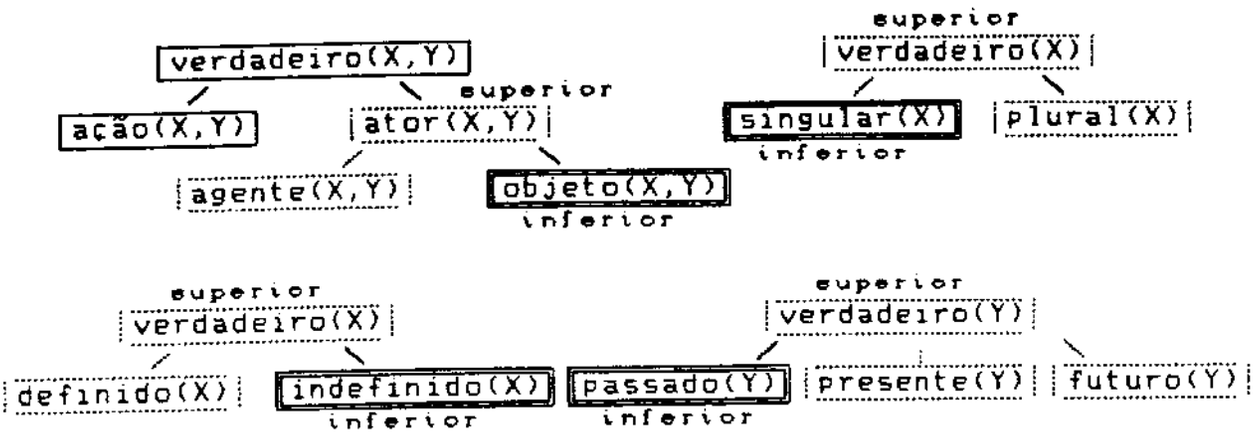


Figura 2.3- Conjunto de árvores de hierarquia para aquisição de linguagem.

Deve ser observado que os nós raízes por serem sempre verdadeiros podem ser omitidos da regra.

Os procedimentos para a manipulação dos marcadores são a *Generalização* e a *Discriminação*.

Generalização

O processo de Generalização requer um contexto de seleção correspondente a aplicação correta da regra e um conjunto de árvores de descritores juntamente com os seus marcadores. Neste processo cada árvore é tratada separadamente e o resultado são novos marcadores inferiores para algumas das árvores. O processo de Generalização é composto dos seguintes passos para cada árvore:

- Avaliar todos os descritores terminais no contexto de seleção.

- Rotular o nó sob o descritor avaliado verdadeiro como nó corrente.

- Fixar na árvore um novo marcador inferior, encontrando o mínimo limite superior entre o nó corrente e o marcador inferior atual.

Se for adotada uma visão específica para transformação da regra, a mudança da posição do marcador inferior modificará de maneira explícita a forma da regra. Caso contrário, a modificação somente elevará o nível mínimo de generalidade potencial da regra sem alterar a sua forma.

Exemplo:

Supondo a regra

descreve(X) → prefixo(X,um)

a qual foi aplicada corretamente no contexto

(cachorro/X,evento1/Y)

e supondo também que os marcadores estão dispostos como mostrado na figura 2.4, as relações terminais que são verdadeiras para o contexto de seleção são:

agente(cachorro,evento1),

singular(cachorro),

indefinido(cachorro),

presente(evento1),

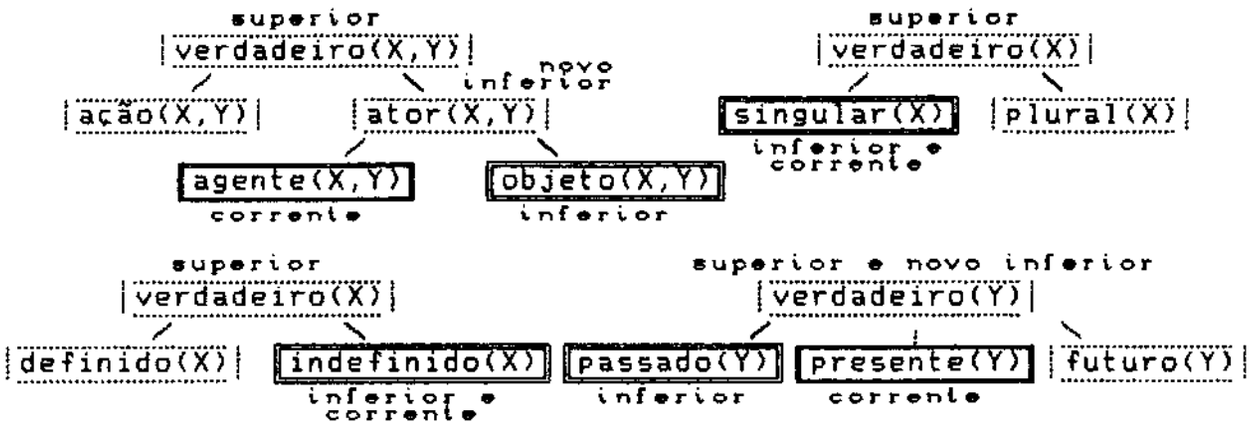


Figura 2.4 - Aplicação da Generalização sobre conjunto de árvores de hierarquia.

Discriminação

O processo de Discriminação usa como entradas um contexto de rejeição correspondente a aplicação incorreta da regra e um conjunto de árvores de descritores referentes a regra. Os seguintes passos são executados para cada árvore:

- Avaliar todos os descritores rotulando os nós terminais da árvore no contexto de rejeição.
- Rotular como nó corrente aquele sob o descritor avaliado como verdadeiro. Este nó estará sempre abaixo do marcador superior.
- Se o nó corrente localizar-se abaixo do marcador inferior, a árvore deve ser ignorada.
- Se o nó corrente localizar-se entre o marcador superior e o marcador inferior a árvore deverá ser considerada.

Um novo marcador superior é fixado de maneira suficiente para que o nó corrente seja excluído. Isto é conseguido fixando o novo marcador superior como um único nó abaixo do nó limite mínimo superior entre o nó corrente e o marcador inferior, que ainda está acima do marcador inferior.

Exemplo:

Supondo a regra

descreve(X) → prefixo(X,um)

a qual foi aplicada incorretamente no contexto de rejeição (persegue/X,evento2/Y), que ocorre, por exemplo, na frase "O cachorro um persegue a bola":

(persegue/X,evento2/Y)

e supondo também que os marcadores estão dispostos como mostrado na figura 2.5, as relações terminais que são verdadeiras no contexto de rejeição são:

ação(persegue,evento2),

singular(persegue),

indefinido(persegue),

presente(evento2).

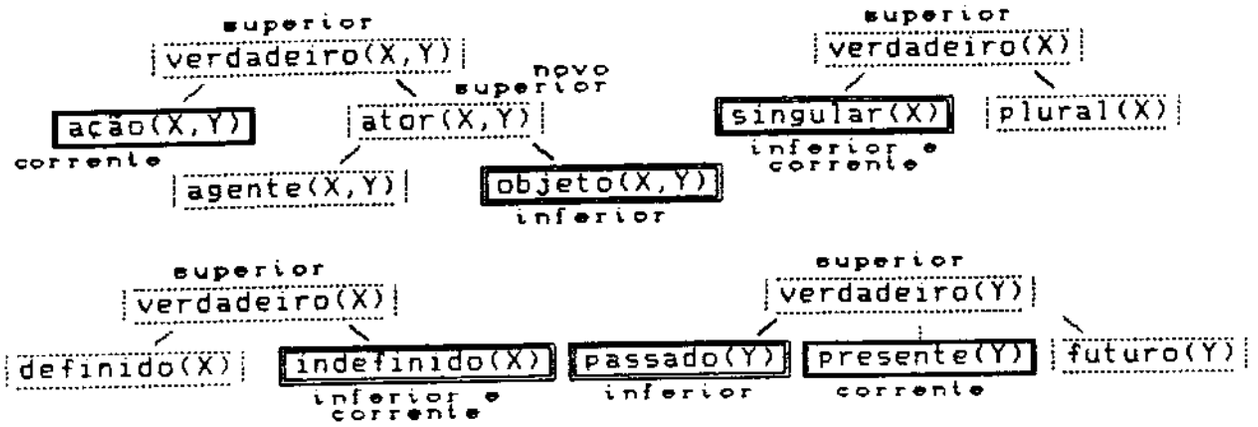


Figura 2.5 - Aplicação da Discriminação sobre um conjunto de árvores hierárquicas.

A nova regra é:

descreve(X),ator(X,Y) → prefixo(X,um).

Podem ocorrer casos em que somente uma árvore é considerada para fixação de um novo marcador superior. Neste caso temos um "*near miss*". Quando ocorrer que mais de uma árvore forem consideradas ,temos um "*far miss*". Neste último caso tem-se várias opções para alteração dos marcadores superior de maneira a descartar um descritor inadequado; porém , somente uma deverá ser efetivada. A árvore escolhida é chamada de *discriminante*. A escolha de um discriminante a ser usado pode ser realizada através das seguintes estratégias:

- busca com "*backtracking*" [22] - Um discriminante qualquer é escolhido; porém, se a regra obtida mostrar-se posteriormente incorreta, é realizado um "*backtracking*" ao ponto de decisão e um outro discriminante é escolhido.
- busca em largura [22]- Criam-se novas regras associadas a cada discriminante.
- opção do professor - O professor (pessoa que orienta o processo de aprendizagem) decide qual discriminante usar.
- opção nula - nada é feito
- opção de prevenção - Os exemplos são apresentados em uma ordem adequada, de maneira que um "*far miss*" não ocorra.

A escolha do novo marcador superior como uma função dos nós correntes e inferior ao invés do superior, garante que o processo de Discriminação alterará o marcador superior no sentido de reduzir a região de indecisão da árvore. A subárvore definida pelo marcador superior sempre conterá a subárvore definida pelo marcador inferior. O procedimento de Discriminação visa eliminar, ou explicitamente (visão geral), ou potencialmente, os descritores que causam erro de controle na regra. Quando a condição da regra é formada como uma conjunção dos descritores sob o marcador superior, a eliminação do descritor avaliado como verdadeiro no contexto de rejeição da regra será suficiente para

que a regra não seja disparada naquele contexto, tornando a regra mais restrita.

O algoritmo de Focalização exige um controle rigoroso na ordem dos exemplos fornecidos para que não se verifiquem situações de inconsistência.

Os problemas de inconsistência surgem quando a ordem dos exemplos apresentados fixa os marcadores de maneira que um próximo exemplo positivo defina nós correntes acima do marcador superior, e dessa forma desabilite-os de contribuirem no processo de generalização, ou um próximo exemplo negativo defina nós correntes abaixo do marcador inferior, inibindo-os de contribuirem na escolha de um discriminante adequado.

A inconsistência é evidência de uma má escolha dos descritores ou da ordem dos exemplos. Os exemplos que provocam inconsistência podem ser simplesmente ignorados, e considerados como dados espúrios. A árvore hierárquica onde ocorreu inconsistência deve ser reorganizada, e acrescida de novos descritores.

2.2.2.2 - Regras de generalização de descrições

2.2.2.2.1 - Regras de generalização seletiva [20]

Nas regras de generalização seletiva o principal ponto de atuação são os domínios de cada descritor usado na descrição do evento observado. O domínio de um descritor determina os objetos no universo do evento para os quais o descritor é válido. Dada a descrição de um evento, a semântica de um descritor determina os níveis de generalização aplicáveis ao domínio do descritor. Um segundo ponto de atuação das regras de generalização seletiva são os próprios descritores da descrição observada, porém, esta atuação das regras se restringe a eliminação de algum descritor que esta presente na descrição inicial do evento. Esta eliminação pode ser explicada quando definimos uma descrição como uma conjunção de descritores. Um determinado objeto sobre o qual um determinado descritor é

aplicável, pode ter como resultado de sua avaliação os valores *V*(verdadeiro) ou *F*(falso), dependendo se o objeto satisfaz ou não as condições impostas pelo descritor. Caso após a generalização as condições impostas pelo descritor sejam tais que todo o seu domínio é aceito, sua avaliação sempre será verdadeira, e o descritor passará a ser um fator dispensável para a diferenciação de eventos que pertencem dos que não pertencem a uma determinada classe.

A seguir estão algumas regras de generalização seletiva. O enunciado das regras segue a notação de APC, e a seguinte simbologia:

$\sim$	negação.
$\wedge$	conjunção (ou produto lógico)
$\vee$	disjunção (ou soma lógica)
$\Rightarrow$	implicação
$\Leftrightarrow$	equivalência lógica
$\mid<$	generalização
$\mid>$	especialização
$\mid=$	reformulação
$::>$	uma implicação ligando uma descrição do conceito ao nome do conceito.
K_i	um predicado especificando o nome de um conceito (uma classe)

• Regra de abandono de condições.

Esta regra é derivada da seguinte tautologia proposicional [3,22].

$$P \wedge Q \Rightarrow P$$

Onde P e Q representam trechos de uma descrição.

O enunciado da regra de generalização seletiva baseada nesta tautologia é:

$$CTX \wedge S ::> K \quad \mid< CTX ::> K$$

onde *CTX* é uma descrição contextual e S é um descritor. Esta regra enuncia que se uma descrição $CTX \wedge S$ é um exemplo positivo de um evento da classe *K*, a descrição mais geral *CTX* também corresponde a uma descrição da classe *K*.

O termo *mais geral* mencionado acima esta relacionado ao número de eventos que satisfazem as condições expressas na descrição. Um menor número de condições propicia um maior número de possíveis eventos que podem ser classificados como pertencentes a classe *K*, e por isso, esta descrição menos restritiva é dita ser *mais geral*, e a descrição obtida após esta transformação corresponde a uma generalização da descrição inicial.

Ex: Dado $P \wedge Q \Rightarrow P$

onde *P* equivale a declaração *objetos redondos* e *Q* equivale a *objetos marrons*, podemos generalizar a descrição para o conceito de *BOLA* como segue:

$$P \wedge Q \ ::) \ BOLA \ |< \ P \ ::) \ BOLA$$

• Transformação de conjunções em disjunções

Esta regra constitui um caso mais geral da regra de abandono de condição. Seu enunciado é:

$$F1 \wedge F2 \ ::) \ K \ |< \ F1 \vee F2 \ ::) \ K$$

onde *F1* e *F2* são trechos de uma descrição. A descrição resultante desta transformação pode ser interpretada como duas descrições exclusivas, $F1 \ ::) \ K$ e $F2 \ ::) \ K$, e correspondem a uma descrição mais geral, obtida pela fragmentação da descrição inicial, e abandono dos trechos complementares.

• Regra de resolução indutiva

Esta regra esta baseada no princípio de resolução lógica [3,22], o qual é enunciado na sua forma proposicional como:

$$(P \Rightarrow F1) \wedge (\neg P \Rightarrow F2) \ |) \ F1 \vee F2$$

onde P é um predicado e $F1$ e $F2$ são fórmulas arbitrárias. A sua utilização no processo de generalização da descrição de um conceito é realizado de acordo com o enunciado abaixo.

$$\left.\begin{array}{l} P \wedge F1 \ ::) \ K \\ \sim P \wedge F2 \ ::) \ K \end{array}\right| \langle \ F1 \vee F2 \ ::) \ K$$

As descrições iniciais são assumidas estarem relacionadas conjuntivamente. O exemplo a seguir ilustra a atuação desta regra. Admita os seguintes símbolos e seus significados:

K significa *João vai ao cinema.*

P significa *João tem companhia.*

$F1$ significa *o filme é bom.*

$F2$ significa *João esta folgado.*

São dados dois exemplos positivos do evento K, que são:

$P \wedge F1$, significa *João tem companhia e o filme é bom.*

$\sim P \wedge F2$, significa *João não tem companhia e João esta folgado.*

A descrição generalizada é $F1 \vee F2$, e corresponde ao seguinte texto.

João vai ao cinema quando o filme é bom ou quando ele está folgado.

* Regra da contra-extensão

Seu enunciado é:

$$\left.\begin{array}{l} CTX1 \wedge [L = R1] \ ::) \ K \\ CTX2 \wedge [L = R2] \ ::) \ \sim K \end{array}\right| \langle \ [L \neq R2] \ ::) \ K$$

onde $CTX1$ e $CTX2$ são descrições contextuais, L é um termo, e $R1$ e $R2$ são conjuntos. Esta regra destaca o termo L como sendo a explicação para que a segunda descrição seja um

exemplo negativo de K. O valor de L é tomado como condição única e discriminante entre o exemplo positivo e negativo. A restrição que L seja diferente de $R2$ na descrição resultante, impõe uma restrição aos valores que L pode assumir. Porém, quanto aos possíveis valores permitidos inicialmente pertencentes ao conjunto $R1$, os novos valores promovem uma generalização na descrição.

***Regra de extensão de referência**

Seu enunciado é:

$$CTX \wedge [L = R1] \;::\rangle\; K \;\big|\langle\; CTX \wedge [L = R2] \;::\rangle\; K$$

$R1$ e $R2$ constituem conjuntos de elementos disjuntos, e observam as seguintes relações:

$R1 \subseteq R2 \subseteq DOM(L)$ onde $DOM(L)$ denota o domínio de L.

Esta regra de generalização atua no sentido de ampliar o número dos possíveis valores assumíveis pelo termo L para a descrição de um evento da classe K. Ao ser definido o descritor L, $DOM(L)$ é também definido; porém, as descrições dos exemplos positivos podem evidenciar um subconjunto de $DOM(L)$ útil na discriminação de eventos que pertencem dos que não pertencem a classe K. $R2$ deve ser um subconjunto de $DOM(L)$, e no máximo será igual a $DOM(L)$. Neste segundo caso, fica claro que a diferenciação entre eventos da classe K e $\tilde{}K$, não esta no valor assumido por L, e neste caso esta condição pode ser descartada da descrição resultante.

***Regra do intervalo fechado [20,33]**

Seu enunciado é:

$$\left.\begin{array}{l} CTX \wedge [L = a] \;::\rangle\; K \\ CTX \wedge [L = b] \;::\rangle\; K \end{array}\right| \langle\; CTX \wedge [L = a..b] \;::\rangle\; K$$

onde L é um descritor linear, a e b são valores

específicos de L. Esta regra cria um intervalo fechado entre os valores a e b sobre o qual o descritor L pode ser considerado válido na caracterização de um evento da classe K. As descrições iniciais são consideradas relacionadas conjuntivamente, e a única diferença entre elas está no valor do descritor linear L.

***Regra de generalização em árvore [20,33]**

Seu enunciado é:

$$\left.\begin{array}{l} CTX \wedge [L = c_1] :: \rangle K \\ CTX \wedge [L = c_2] :: \rangle K \\ \quad\vdots \\ CTX \wedge [L = c_i] :: \rangle K \end{array}\right| \langle\, CTX \wedge [L = s] :: \rangle K$$

onde L é um descritor estruturado, e s representa o mínimo antecessor comum entre os nós $c_1, c_2, \ldots, c_i$. O domínio de um descritor estruturado está organizado em uma estrutura de árvore, que define uma ordem hierárquica de generalização entre os seus componentes. Esta regra de generalização utiliza-se desta organização para gerar uma elevação do nível máximo de hierarquia dos elementos que atribuem um valor para L.

***Regra de transformação de constantes em variáveis [20]**

Seu enunciado é:

$$\left.\begin{array}{l} F[c_1] \\ F[c_2] \\ \\ F[c_i] \end{array}\right| \langle\, \forall\, v\ , F[v]$$

onde $F[v]$ é uma descrição dependente da variável v, e c_i para i maior ou igual a 1, são constantes. Se a descrição $F[v]$ é verdadeira para v igual a todo c_i, a generalização afirma que $F[v]$ é verdadeira para qualquer valor de v. Esta regra corresponde ao inverso da regra de inferência lógica conhecida como especialização universal [22].

2.2.2.2.2 - Regras de Generalização Construtiva

As regras de generalização construtivas são caracterizadas por produzirem uma descrição geral, contendo descritores que não estão presentes na descrição inicial. A presença destes novos descritores representa uma transformação no espaço de representação original. Os novos descritores acrescentados na afirmação indutiva, relacionam-se a alguns descritores na descrição inicial. Este relacionamento é tido como conhecido pelo sistema de aprendizagem.

A regra de generalização construtiva pode ser enunciada como segue:

$$\begin{array}{ll} 1) & CTX \wedge F1 \ ::) \ K \\ 2) & F1 \Rightarrow F2 \end{array} \Big| \ \langle \ CTX \ \wedge F2 \ ::) \ K$$

onde 1) é o texto descritivo inicial, e 2) define uma relação de implicação conhecida a priori entre o conceito $F1$ e $F2$. $F2$ nesta implicação pode ser obtido por meio da aplicação de alguma função empírica sobre $F1$, que resulte em novos descritores. A relação entre $F1$ e $F2$ pode também ter sido aprendida anteriormente, ou pode ter sido fornecida ao sistema como parte do conhecimento inicial.

Ex:

Dada as expressões:

1) $\exists \ P \ [cor(P) = preta] \ [largura(P) \wedge comprimento(P) = grande] \ ::) \ K$

representando que um objeto pertencente a classe K (um quadro negro) apresenta cor preta, largura grande e comprimento grande.

2) $\forall \ P \ [largura(P) \wedge comprimento(P) = grande] \Rightarrow [área(P) = grande]$

que enuncia um conhecimento inicial representando que se um objeto apresenta largura grande e comprimento grande, então ele apresentará uma área grande.

A generalização obtida destas duas expressões acima é:

$\exists \ P \ , [cor(P) = preta] \ [área(P) = grande] \ ::) \ K$

Esta expressão nos diz que um objeto que apresenta cor preta e área grande pertence a classe *K*.

Os descritores *cor*, *largura* e *comprimento* inicialmente usados neste exemplo, definem um conjunto de expressões possíveis das quais o exemplo faz parte. Este conjunto de expressões corresponde a uma linguagem de descrições e expressam estados do universo observado, de acordo com o significado pretendido dos descritores. A alteração na descrição baseada nestes descritores para uma descrição baseada nos descritores *cor* e *área* representa uma alteração na linguagem de descrições e, consequentemente, da maneira como o estado do universo é descrito, pois o significado do descritor *área* expressa novos estados do universo. Esta mudança no conjunto de descritores representa uma transformação no espaço de representação original.

2.3 - Considerações

Considerando-se a teoria cognitiva [27] de aprendizagem supôs-se que a aprendizagem é um processo de criação e manipulação de modelos. A composição de um modelo se dá por meio da definição de uma linguagem de representação, e esta linguagem define as técnicas de manipulação aplicáveis ao modelo. As regras de formação de sentenças de uma linguagem formam uma gramática para aquela linguagem. Das gramáticas apresentadas observam-se algumas características interessantes das linguagens lógicas em relação as linguagens bidimensionais, nos aspectos de representação e manipulação de modelos. As linguagens bidimensionais favorecem a representação, mas não a manipulação. As regras destas linguagens são específicas para representação e análise sintática de padrões, e os processos de manipulação das mesmas visando a aprendizagem não estão ainda sedimentados [8,9].

A GCD é uma das linguagens lógicas e apresenta apresenta facilidades de representação e manipulação. O GCD herda o poder de representação das GLC e a capacidade de manipulação

destas regras deriva do formalismo de cláusulas definidas. O GCD desta forma é uma maneira de se expressar modelos para padrões com maior versatilidade e flexibilidade, devido a sua generalidade e facilidade de manipulação de suas regras. Outra vantagem do uso de linguagens derivadas da lógica é a facilidade de interpretação do conteúdo aprendido, expresso por estas gramáticas, o que o torna transparente ao usuário, e facilita o processo de correção do sistema de aprendizagem durante a fase de desenvolvimento.

Considerando as linguagens derivadas da lógica como as mais adequadas para a representação dos padrões, foram apresentadas algumas técnicas de manipulação de expressões sobre estas linguagens, visando a aprendizagem. Pode ser observado na literatura uma diversidade de formalismos em sistemas de aprendizagem [1,12-15,20,25,26,29-34], o que evidencia uma base teórica ainda não sedimentada no entendimento destes processos. As técnicas de generalização são análogas a um algoritmo para criação de suposições a partir de eventos observados. Neste sentido não é possível a preservação de verdades no processo de generalização, ou seja, não é garantido que a generalização de descrições de eventos sejam válidas para outros eventos além daqueles que serviram de base para aquela generalização.

3 - IMPLEMENTAÇÃO

3.1 - Introdução

O trabalho objetiva o reconhecimento de topologias funcionais de circuitos eletrônicos e a criação de regras de reconhecimento destas topologias.

O problema de criação de regras de reconhecimento de topologias de circuitos é visto como um problema de aprendizagem, e o reconhecimento de topologias, como utilização das regras aprendidas. A aprendizagem utiliza-se do processo de generalização de descrições representando o circuito a ser aprendido. Estas descrições e as suas generalizações, as quais correspondem a regras de reconhecimento do circuito, são expressas por meio da linguagem PROLOG. A linguagem PROLOG [17] foi escolhida por ser ela um linguagem lógica, com facilidades de manipulação simbólica. As regras de generalização sobre descrições em APC podem ser facilmente aplicáveis a cláusulas da linguagem PROLOG, devido as suas semelhanças. As regras de reconhecimento aprendidas correspondem a cláusulas da linguagem PROLOG, e formam um conjunto intelegível de expressões. O conjunto de regras aprendidas constituem um programa PROLOG, o qual pode ser utilizado separadamente do sistema de aprendizagem, sendo possível a sua compilação, para geração de um programa eficiente no reconhecimento de circuitos.

3.2 - Restrições

Dentro do problema de aprendizagem pode-se classificar o sistema desenvolvido como baseado no processo de aprendizagem a partir de exemplos positivos. A impossibilidade do uso de exemplos negativos se deve ao número indeterminado de tais exemplos. Uma vez escolhido um exemplo positivo e expressa a sua topologia, todas as topologias diferentes daquele exemplo podem

ser encaradas como seu exemplo negativo. As topologias funcionais de circuitos eletrônicos usadas como exemplos positivos caracterizam a função. Este tipo de descrição é conhecido como *descrição característica* [20]. O processo de criação de regras é incremental, pois não é possível dispor de todas as topologias de circuitos que possuem determinada função de uma só vez como entrada do sistema.

No desenvolvimento do sistema algumas restrições são impostas, com vistas a delimitação do seu escopo. Estas restrições são as seguintes:

*Utilizam-se como dados de entrada do sistema, informações relativas a uma representação do circuito eletrônico a ser reconhecido, sem levar em consideração os valores dos seus componentes contituintes, enfatizando-se assim a estrutura do circuito.

*As características dos sinais elétricos incidindo sobre os terminais de determinado componente não são considerados, com exceção dos sinais de alimentação do circuito.

*Como consequência das duas primeiras restrições, observa-se que os circuitos adequados ao tratamento pelo sistema são aqueles cujas funções são caracterizadas pelo processamento de sinais, possuindo terminais de entrada, saida e alimentação do circuito distintos e definidos. Isto permite que as regras de reconhecimento sejam livres de contexto [7,28].

As restrições impostas visam definir um nível de detalhamento adequado sobre as descrições caracterizando determinado circuito. A representação do circuito é fornecida ao sistema no formato conhecido como "*netlist*", o qual constitui a descrição do esquemático do circuito. Pode ser observado que o próprio esquemático descreve somente as conexões mais significativas do circuito, constituindo-se numa abstração do circuito real que representa. No esquemático de um circuito toda

informação que não está explicitamente declarada não deve ser considerada, ou seja, é assumido um mundo fechado sobre as informações explicitamente contidas no esquemático do circuito [28]. A representação interna do circuito, que é obtida a partir do *"netlist"* fornecido como entrada do sistema, é restrita somente a informações estruturais. Tais informações serão a descrição inicial do circuito e sofrem transformações de generalização para formarem a regra de reconhecimento do circuito. A segunda restrição sobre o sistema visa limitar a generalidade das regras de reconhecimento, uma vez que os sinais elétricos entre subcomponentes funcionais do circuito podem definir o caracter funcional destas subcomponentes. Ou seja, a característica dos sinais elétricos entre subcomponentes conectados de um circuito servem como informações contextuais definindo a função destes subcomponentes. Apesar da possibilidade da incorporação destas características contextuais às regras, preferiu-se não fazê-lo, visando uma menor complexidade das regras, e consequentemente dos procedimentos de aprendizagem. Um exemplo desta dependência contextual é observado no seguinte exemplo.

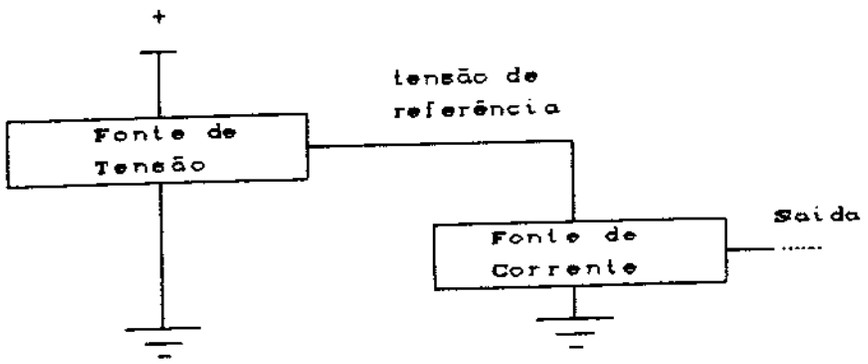


Figura 3.1 - Exemplo de reconhecimento dependente de características dos nós.

Para que seja reconhecida uma fonte de corrente é necessário que além do reconhecimento estrutural exista uma tensão de referência entre seus terminais.

3.3 - Sistema

Os diversos componentes do sistema estão representados no diagrama abaixo:

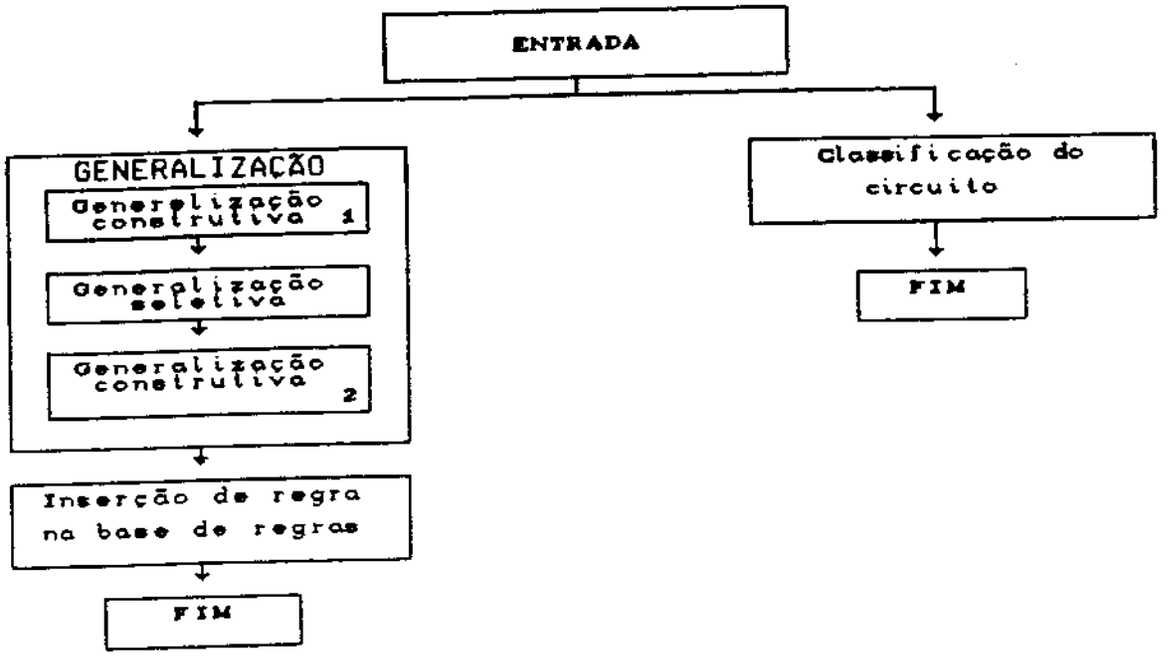


Figura 3.2 - Estrutura do sistema desenvolvido

A seguir cada bloco é descrito.

3.3.1 - Entrada

O bloco de entrada é constituido por três outros blocos, conforme o diagrama abaixo.

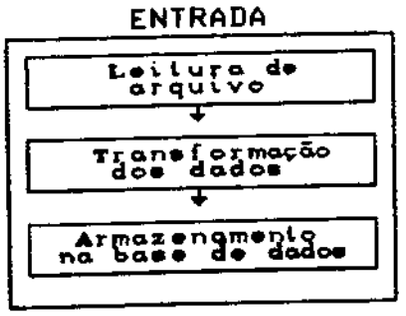


Figura 3.3 - Bloco de entrada

O dado de entrada corresponde a um arquivo em formato de entrada para o programa PSPICE [18]. Cada componente do circuito que é identificado no arquivo, é transformado em um descritor equivalente que o represente, e este descritor é por fim armazenado na base de dados. No fim do processo de entrada o conteúdo da base de dados corresponde a uma descrição do circuito, que serve como exemplo positivo. Para definição dos descritores para expressão das estruturas do circuito observou-se que o mesmo pode ser visto como um conjunto de relacionamentos entre seus nós e componentes. Seguindo esta observação e limitando-se os componentes aos tipos básicos definidos como entrada de um arquivo PSPICE, foram definidos os seguintes descritores:

diodo(Nome, No1, No2)

Indica a existência do diodo *Nome* entre os nós *No1* (positivo) e *No2* (negativo).

resistor(Nome, No1, No2)

Indica a existência do resistor *Nome* entre os nós *No1* e *No2*.

transistor(Nome, No1, No2, No3)

Indica a existência do transistor *Nome*, sendo o seu coletor conectado a *No1*, sua base ao *No2*, e seu emissor ao *No3*.

trans_BIPOC Nome, No1, No2, No3) ou *trans_BIPOC Nome, No1, No2, No3, No4)*

Indica a existência do transistor bipolar *Nome*, sendo seu terminal de coletor conectado ao *No1*, sua base ao *No2*, seu emissor ao nó *No3*, e opcionalmente o substrato esta ligado ao *No4*.

trans_JFETC Nome, No1, No2, No3)

Indica a existência do transistor de junção FET *Nome*, sendo seu dreno conectado ao *No1*, sua porta ao *No2* e sua fonte ao *No3*.

trans_MOSFETC Nome, No1, No2, No3) ou
trans_MOSFETC Nome, No1, No2, No3, No4)

Indica a existência do transistor MOSFET *Nome*, sendo seu dreno conectado a *No1*, sua porta a *No2* e sua fonte a *No3*, e opcionalmente tendo o seu substrato conectado ao *No4*.

A transformação da notação usada no arquivo de entrada do PSPICE, para a forma de descritores é imediata, bem como o seu armazenamento na base de dados.

3.3.2 - GENERALIZAÇÃO

Generalização Construtiva

Os blocos de generalização podem ser esquematicamente dispostos como:

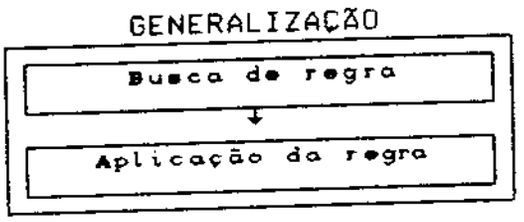


Figura 3.4 - Bloco de generalização

A regra de generalização construtiva é usada no processo de generalização das descrições armazenadas. Esta

escolha propicia que as regras de reconhecimento resultantes da aplicação do processo de generalização incorporem o reconhecimento de subestruturas de maneira hierárquica. Usando-se este processo pode-se obter regras com características interessantes. A primeira característica é derivada da própria definição de generalização construtiva, na qual um relacionamento previamente conhecido entre descritores é utilizado na generalização de uma descrição. No sistema as regras de reconhecimento anteriormente aprendidas são adotadas como expressões para este relacionamento. Desta maneira as regras já aprendidas são incorporadas ao corpo das regras a serem generalizadas, resultando em um conjunto de regras compacto, onde as regras de reconhecimento de circuito estão definidas de maneira recursiva. As regras definidas desta forma, utilizando no seu corpo chamadas de definições de regras já definidas anteriormente, propiciam um processo de análise que destaca a maneira hierárquica na qual os subcomponentes do circuito analisado estão dispostos.

No esquema representativo do sistema podemos observar dois blocos de generalização, que diferem nos seus argumentos. O primeiro bloco de generalização construtiva trata especificamente da generalização da descrição de um circuito, visando a criação de uma regra específica para o seu reconhecimento. As regras aprendidas são usadas de maneira exaustiva, sobre a descrição do circuito. A descrição resultante deste processo corresponde a uma instância da regra de reconhecimento definitiva. Após a generalização seletiva, a regra definitiva é formada, a qual é armazenada na base de regras.

Esquematicamente este bloco pode ser observado como:

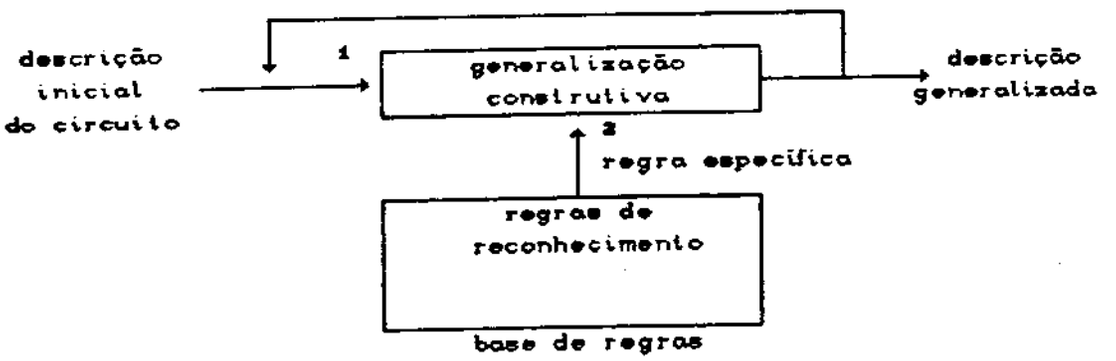


Figura 3.5 - Esquema do bloco de generalização

O texto descritivo inicial a ser generalizado é fornecido a entrada 1 do procedimento de generalização, e o relacionamento previamente conhecido entre descritores, que neste caso é uma regra escolhida da base de regras, é fornecido a entrada 2. Para a escolha da regra fornecida na entrada 2, usou-se algum conhecimento inicial, visando diminuir o número de tentativas de generalização desnecessárias. Observando-se o enunciado da generalização construtiva (pag.72) pode ser observado que alguma instância de F1 deve estar contida na descrição 1). Deste fato pode ser concluido que o número de descritores em 1) deve ser maior que o número de descritores em F1 para que seja aplicável o processo de generalização construtiva. Se formos mais específicos podemos observar que para que seja possível a aplicação do processo de generalização construtiva, o número de descritores de 1) deve ser maior que o número de descritores em F1 para cada tipo de descritor contido na expressão de F1. Como exemplo pode-se supor que na descrição 1) possuimos 5 descritores sendo dois relativos a transistores bipolares, um a diodo, e dois a resistores, e F1 possui uma expressão contendo dois descritores relativos a diodos. Apesar da descrição 1) possuir um maior número de componentes que a expressão F1, em termos específicos F1 não generaliza 1) devido a seu maior número de descritores relativos a diodos. A utilização deste mecanismo de discriminação de regras, não deve ser encarado como uma heurística [22], uma vez que se trata de um conhecimento

exato a respeito das descrições.

Este mecanismo descarta algumas buscas desnecessárias, mas não todas. Mesmo as expressões de Fi que obedecem ao critério estabelecido podem não produzir generalizações da descrição inicial; além disto nada pode ser dito com base neste critério, a respeito das generalizações obtidas. Isto é, as generalizações obtidas não podem ser classificadas como as melhores ou piores que podem ser obtidas. A aplicação das regras de reconhecimento da base de regras sobre a descrição do circuito exemplo apresentado gera, de acordo com a regra de generalização construtiva, um conjunto de descrições generalizadas que correspondem às regras de reconhecimento intermediárias. Uma destas regras de reconhecimento intermediárias é escolhida para a próxima fase de generalização. Este processo para somente quando a regra de reconhecimento intermediária obtida não puder mais ser generalizada. O processo de escolha das regras de reconhecimento intermediárias funciona como uma busca em profundidade [22,33], e associada a esta busca foi adotada uma heurística que visa dirigir o processo de busca para obtenção de expressões descritivas generalizadas com um formato específico. As regras aprendidas pelo sistema devem, além de reconhecer o circuito eletrônico, propiciar a explicitação das subcomponentes funcionais do circuito durante o processo de análise. Esta explicitação corresponde a estrutura em árvore gerada pela análise, a qual descreve hierarquicamente estas subcomponentes.

Para que exista um maior equilíbrio na árvore resultante da análise, adotou-se que cada regra intermediária deve possuir o menor número possível de terminais que são representados pelos descritores simples, e um maior número de não terminais, ou seja, representação de subcomponentes funcionais do circuito. As regras generalizadas que possuirem um maior número de descritores simples são escolhidas como as generalizações de entrada para o próximo passo no processo de generalização, visando desta forma diminuir o seu número de descritores simples. Esta medida de adequabilidade da regra obtida corresponde a uma função local e pode levar a soluções adequadas locais para o

problema, ao invés de um solução global [22,33].

O valor encontrado para o número de descritores é utilizado de duas maneiras distintas durante a execução do primeiro bloco de generalização. A primeira maneira ocorre quando este valor é utilizado na exclusão de algumas regras da base de regras para aplicação sobre o exemplo. Após a exclusão de algumas das regras, o restante das regras é aplicado sobre o exemplo gerando como resultado um conjunto de descrições generalizadas. Cada descrição generalizada deste conjunto é o resultado da aplicação de cada uma das regras da base de regras sobre o exemplo apresentado. Deste conjunto de descrições somente uma é selecionada para o próximo passo de generalização. Esta escolha é feita também com base no número de descritores de cada uma destas descrições generalizadas, e constitui uma utilização heurística deste número.

Este primeiro bloco de generalizações pode ser visto como um processo de generalização *"bottom-up"* [20], onde as descrições mais gerais são obtidas gradualmente a partir de generalizações menos gerais e de regras na base de regras. Isto é mostrado no seguinte diagrama:

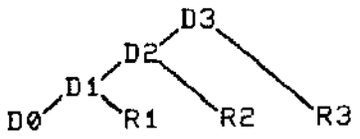


Figura 3.6 - Diagrama representando o processo de generalização "bottom-up".

D3, D2, D1 e D0 são descrições onde Di é mais geral que Di-1 para i = 1 a 3, D0 é a descrição inicial, D3 é a descrição generalizada final, e R1, R2, R3 são regras da base de regras.

O bloco 2 de generalização construtiva atua sobre as regras na base de regras, usando a nova regra aprendida como o texto descrevendo o relacionamento entre descritores. A estrutura deste bloco pode ser descrita pelo seguinte esquema.

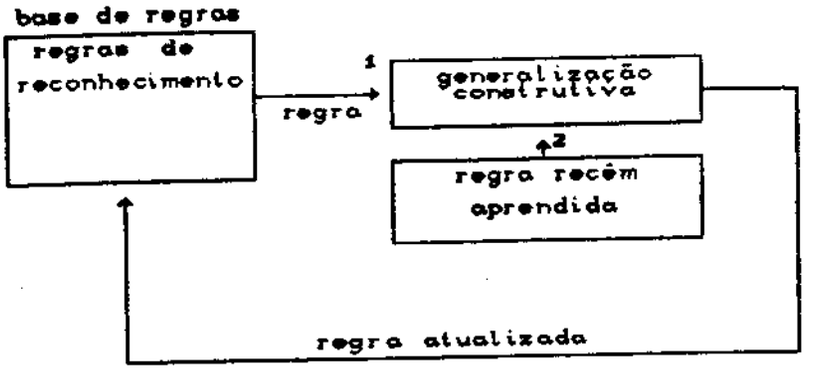


Figura 3.7 - Esquema do processo de atualização de regras na base de regras.

Esta segunda generalização construtiva visa atualizar o conteúdo da base de regras, com base na regra recém aprendida. Todas as regras na base de regras são processadas, com exceção das regras que atuaram de maneira efetiva durante o processo de criação da nova regra. Levando-se em consideração a notação da generalização construtiva (pag. 72) fixamos o texto em 2 como sendo o relacionamento entre os descritores, o qual é expresso pela regra aprendida, e o texto 1 é representado por cada regra na base de regras a ser atualizada. Os dois blocos 1 e 2 de generalização construtiva atuam sobre a forma final da regra de reconhecimento aprendida e na base de regras, a nível de composição de descritores.

Generalização seletiva

O processo de generalização seletiva atua sobre cada descritor da regra, generalizando os objetos sobre os quais o descritor é válido. Para a definição dos descritores podemos observar que os seus domínios são os nós e os componentes de um determinado circuito. Cada componente ou nó de um circuito é uma entidade única e distinta das demais. O processo de generalização deve preservar a identidade de cada nó ou componente do circuito.

Através do mecanismo de generalização seletiva, transformando constantes em variáveis [6], podemos preservar esta identidade dos nós e componentes de maneira relativa, onde as variáveis descrevem elementos particulares, porém indefinidos, denotando-os nas relações (descritores) descrevendo o circuito. Deve ser notado que a adoção deste tipo de generalização impõe algumas restrições ao processo de unificação [3,22]. Cada variável representa um elemento distinto, nó ou componente, e disto resulta que duas variáveis distintas não devem ser instanciadas a um mesmo nó ou componente. Isto pode ser exemplificado através do seguinte exemplo. São dados o descritor *trans_BIPOC2,2,1,q1>*, o qual pode ser interpretado como *"existe um transistor bipolar q1 com coletor ligado ao nó 2, base ligada ao nó 2, e emissor ligado ao nó 1"*, (figura 3.8a),e o descritor generalizado *trans_BIPOCA,B,C,D>*, o qual é interpretado como *"existe um transistor bipolar A, com coletor ligado ao nó B, base ligada ao nó C, e emissor ligado ao nó D"*, (figura 3.8b).

a b

Figura3.8 - Representação de dois transistores: a-transistor eespecifico; b-transistor generalizado.

Pode-se observar que os dois descritores não podem descrever um mesmo elemento, ou seja, não devem unificar, uma vez que *trans_BIPOC2,2,1,q1>* não é uma instância de *trans_BIPOCA,B,C,D>*. Usando a notação PROLOG, pode-se expressar esta característica restritiva do descritor no programa pelo seguinte texto:

...,*transistor(A,B,C,D)*, *B* \= *C*, *B* \= *D*, *C* \= *D*,...

onde o simbolo \= significa não unifica.

O bloco de generalização seletiva pode ser descrito como:

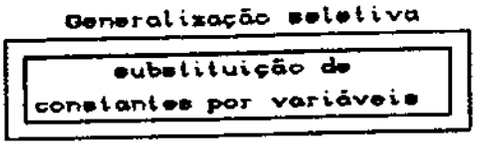


Figura 3.9 - Bloco de generalização seletiva

A aplicação da generalização seletiva se dá substituindo-se todas as constantes da regra resultante do bloco 1 por variáveis.

3.3.3 - Inserção de Regra

Consiste simplesmente da inserção da regra na base de regras. Esta ação, porém, ocorre somente se nenhuma irregularidade foi encontrada no processo de generalização. Uma irregularidade ocorre quando se tenta aprender regras já aprendidas anteriormente pelo sistema. Esta situação pode ser detectada durante o processo de generalização construtiva (pag. 72), quando o corpo F1 corresponde a uma generalização da regra a ser aprendida, ou seja, CTX é vazio. Alguns dados são solicitados ao usuário a respeito do circuito de entrada, no início do processo de aprendizagem. Depois que a regra é criada, ela é apresentada ao usuário, o qual pode aceitá-la ou rejeitá-la. Isto previne a inclusão na base de regras de regras incorretas, que podem ser criadas quando o usuário fornece dados inadequados. Caso a regra seja rejeitada todas as transformações na base de regras associadas a esta regras devem ser desfeitas, retornando-a ao seu estado inicial antes da aprendizagem .

3.3.4 - Classificação

O processo de classificação de um circuito consiste na aplicação das regras aprendidas, usando como entrada a descrição do circuito na base de dados. As regras correspondem a cláusulas da linguagem PROLOG, de maneira que o processo de classificação consiste simplesmente da execução das regras aprendidas no ambiente PROLOG. As cláusulas então podem ser interpretadas como cláusulas de uma GCD, e possuem as propriedades descritas no capitulo 2. Uma destas propriedades é a possibilidade de obtenção de múltiplas soluções para a análise, através do mecanismo de *"backtracking"*. As regras também podem ser utilizadas na identificação de subcircuitos específicos. O resultado da análise é uma estrutura em árvore descrevendo a classificação do circuito global, que enumera todas as subtopologias funcionais do circuito e os seus elementos terminais, ou seja, os componentes eletrônicos que o formam. A classificação de um circuito pode também ser parcial, com a identificação de subtopologias parciais do circuito.

4 - RESULTADOS

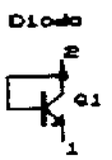
4.1 - Criação de Regras

Como uma ilustração do funcionamento do sistema durante o procedimento de criação de regras para reconhecimento a partir de exemplos, podemos considerar três situações principais no estado do sistema:

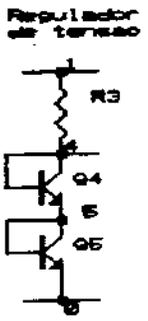
- O circuito apresentado como exemplo corresponde à primeira regra na base de regras.
- O circuito apresentado como exemplo pode ser visto como uma composição de circuitos mais simples, cujas regras se encontram na base de regras.
- O circuito apresentado como exemplo pode ser visto como um subcomponente de estruturas mais complexas, cujas regras se encontram na base de regras.

Para ilustrar estas três situações, consideramos como exemplo os esquemáticos e correspondentes *netlists* em formato PSPICE [18] abaixo:

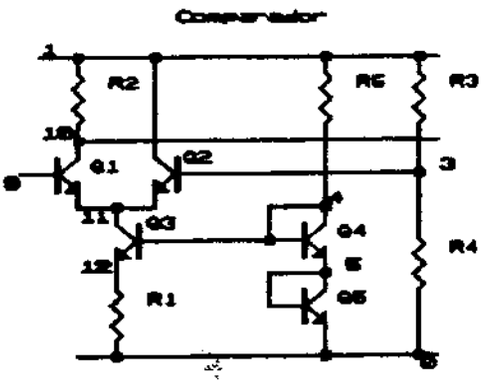
```
TRANSISTOR COMO DIODO
* TERMINAIS = 1,2
Q1 2 2 1 M
.END
```




```
CIRCUITO REGULADOR DE TENSAO
* IN = 0,1
* OUT = 4
* POT = 0,1
R1 1 4 M
Q1 4 4 5 M
Q2 5 5 0 M
.END
```



```
COMPARADOR
*IN = 9
* OUT = 10
* POT = 0,1
Q1 10 9 11 M
Q2 1 3 11 M
Q3 11 4 12 M
Q4 4 4 5 M
Q5 5 5 0 M
R1 12 0 M
R2 1 10 M
R3 1 3 M
R4 3 0 M
R5 1 4 M
.END
```



A seguir cada situação é ilustrada, através da apresentação dos resultados intermediários obtidos pela execução de cada bloco principal do sistema e as suas consequências na base de regras.

***Primeira situação:**

O seguinte *netlist* do circuito exemplo é apresentada (lido do arquivo especificado pelo usuário).

```
CIRCUITO REGULADOR DE TENSAO
* IN = 0,1
* OUT = 4
* POT = 0,1
R1 1 4 M
Q1 4 4 5 M
Q2 5 5 0 M
.END
```

1)Bloco de entrada.

As seguintes informações são obtidas do usuário:

Nós de alimentação do circuito, nome do circuito, terminais de entrada do circuito, terminais de saída do circuito, terminais para conexões externas .

O bloco de entrada transforma o *netlist* numa notação adequada ao tratamento pelo PROLOG. O restante das informações colhidas do usuário serão usadas posteriormente.

Segue-se a notação em PROLOG interna ao sistema obtida da *netlist* apresentada:

```
resistor(s,r1,1,4)
tran_BIPO(s,q1,4,4,5)
tran_BIPO(s,q2,5,5,0)
```

2)Bloco de generalização construtiva 1.

Este bloco não é utilizado pois não existe nenhuma regra na base de regras.

3)Bloco de generalização seletiva.

Com a descrição do circuito em sua forma *netlist* e as informações iniciais obtidas do usuário é criada a seguinte regra em notação PROLOG do sistema.

```
circuito(s,1,regulador_de_tensao,[regulador_de_tensao,A,B,C],
entrada(D,E),saida(F),potencia(D,E),conexoes(D,E,F)):-
   ? tran_BIPO(s,A,G,G,D),
    ? tran_BIPO(s,B,F,F,G),
    ( ? resistor(s,C,E,F) ; ? resistor(s,C,F,E)),
    restricoes([G],[D,E]).
```

A regra acima ilustra a estrutura típica das regras criadas pelo sistema. O trecho *circuito(s,1,regulador_de_tensao, [regulador_de_tensao,A,B,C],entrada(D,E),saida(F),potencia(D,E), conexoes(D,E,F))* corresponde a cabeça da cláusula. O argumento 1 (*s*) é um código de controle para o sistema. O argumento 2 (*1*) é o número da regra, sendo que cada regra possui o seu número. O argumento 3 (*regulador_de_tensao*) refere-se ao nome da estrutura correspondente à regra, podendo haver mais de uma regra com um

mesmo nome. O argumento 4 (*[regulador_de_tensao,A,B,C]*) é um argumento para retorno de resultados, e caso a regra seja satisfeita pela estrutura, as variáveis serão instanciadas com os nomes dos subcomponentes que compõem o *regulador_de_tensao*. Os argumentos 5 (*entrada(D,E)*), 6 (*saida(F)*) , 7 (*potencia(D,E)*) e 8 (*conexoes(D,E,F)*) são também argumentos para o retorno de resultados, e caso a regra seja satisfeita pela estrutura, as variáveis serão instanciadas com os valores dos nós correspondentes a entrada, saida, alimentação e conexões externas da estrutura reconhecida.

Os elementos *(tran_BIPO(s,A,G,G,D)*, *(tran_BIPO(s,B,F,F,G)* e *((resistor(s,C,E,F)* ; *(resistor(s,C,F,E))*, representam os elementos terminais da regra, e durante o processo de análise detectam a existência dos componentes eletrônicos básicos formando a estrutura analisada.

O elemento *restricoes([G],[D,E])*, no corpo da regra, é um predicado que verifica se existem outros componentes eletrônicos terminais conectado nos nós internos de um circuito, além daqueles declarados na regra de reconhecimento do circuito. Este elemento pode ser ativada ou desativada, alterando desta maneira o grau de reconhecimento de estruturas pela regra. Quando esta componente do corpo da regra está ativada, a estrutura correspondente a regra será reconhecida se for constituida exclusivamente pelos elementos descritos pelos terminais e não terminais da regra, ou seja, entre os nós internos do circuito (todos que não pertencem ou a entrada, ou a saida, ou a alimentação do circuito) não existirem outros elementos além daqueles anunciados pela regra. Quando a componente *restricoes([G],[D,E])* está desativada, a estrutura é reconhecida pela regra independente da existência de outros componentes ligados aos seus nós internos.

4)Bloco de generalização construtiva 2.

Nesta primeira situação é considerado que não existem regras na base de regras; com isto, o processo de generalização construtiva 2 não é possível.

5)Resultado

A regra resultante do bloco de generalização seletiva corresponde a versão final da regra, a qual será armazenada na base de regras, constituindo-se na sua primeira regra. A base de regras após a aquisição desta primeira regra se encontra na forma:

BASE DE REGRAS

```
circuito(s,1,regulador_de_tensao,[regulador_de_tensao,A,B,C],
entrada(D,E),saida(F),potencia(D,E),conexoes(D,E,F)):-
    ? tran_BIPO(s,A,G,G,D),
    ? tran_BIPO(s,B,F,F,G),
    ( ? resistor(s,C,E,F) ; ? resistor(s,C,F,E)),
    restricoes([G],[D,E]).
```

• **Segunda situação:**

Netlist apresentada ao circuito:

```
COMPARADOR
*IN = 9
* OUT = 10
* POT = 0,1
Q1 10 9 11 M
Q2 1 3 11 M
Q3 11 4 12 M
Q4 4 4 5 M
Q5 5 5 0 M
R1 12 0 M
R2 1 10 M
R3 1 3 M
R4 3 0 M
R5 1 4 M
.END
```

1)Bloco de entrada

Forma interna em PROLOG, obtida a partir do *netlist* fornecido:

```
tran_BIPOC(s,q1,10,9,11)
tran_BIPOC(s,q2,1,3,11)
tran_BIPOC(s,q3,11,4,12)
tran_BIPOC(s,q4,4,4,5)
tran_BIPOC(s,q5,5,5,0)
resistor(s,r1,12,0)
resistor(s,r2,1,10)
resistor(s,r3,1,3)
resistor(s,r4,3,0)
resistor(s,r5,1,4)
```

2)Bloco de generalização construtiva 1

A regra na base de regras pode ser utilizada no processo de generalização construtiva sobre o circuito apresentado como exemplo, dando como resultado a seguinte regra específica.

```
circuito(s,_69DC,_69D8,[_69D8,$[regulador_de_tensao,q5,q4,r5]$,
$q1$,$q2$,$q3$,$r1$,$r2$,$r3$,$r4$],entrada(9),saida(10),
potencia(0,1),conexoes(0,1,9,10)):-
     ? circuito(s,_9898,regulador_de_tensao,
[regulador_de_tensao,q5,q4,r5],entrada(0,1),saida(4),potencia(0,1
),conexoes(0,1,4)),
     ? tran_BIPOC(s,q1,10,9,11)),
     ? tran_BIPOC(s,q2,1,3,11)),
     ? tran_BIPOC(s,q3,11,4,12)),
     ( ? resistor(s,r1,12,0) ; ? resistor(s,r1,0,12)),
     ( ? resistor(s,r2,1,10) ; ? resistor(s,r2,10,1)),
     ( ? resistor(s,r3,1,3) ; ? resistor(s,r3,3,1)),
     ( ? resistor(s,r4,3,0) ; ? resistor(s,r4,0,3)),
     restricoes([3,4,11,12],[0,1]).
```

3)Bloco de generalização seletiva

Transforma a regra específica numa forma mais geral. Nesta segunda situação pode ser observado um novo elemento no corpo da regra,ou seja, *fcircuito(S,Q,regulador_de_tensao,A, entrada(K,L),saida(O),potencia(K,L),conexoes(K,L,O))*. Este elemento expressa a necessidade de reconhecimento de uma subestrutura, cuja regra já se encontra armazenada na base de regras, e funciona como uma componente não terminal do corpo da regra.

Após a execução deste bloco, a regra formada constitui o resultado final a ser acrescido na base de regras, a qual fica da seguinte forma:

BASE DE REGRAS

```
circuito(s,1,regulador_de_tensao,[regulador_de_tensao,A,B,C],
entrada(D,E),saida(F),potencia(D,E),conexoes(D,E,F)):-
    ? tran_BIPO(s,A,G,G,D),
    ? tran_BIPO(s,B,F,F,G),
    ( ? resistor(s,C,E,F)
    ; ? resistor(s,C,F,E)
    ),
    restricoes([G],[D,E]).
circuito(s,2,comparador,[comparador,A,B,C,D,E,F,G,H],
entrada(I),saida(J),potencia(K,L),conexoes(K,L,I,J)):-
    ? tran_BIPO(s,B,J,I,M),
    ? tran_BIPO(s,C,L,N,M),
    ? tran_BIPO(s,D,M,O,P),
    ( ? resistor(s,E,P,K)
    ; ? resistor(s,E,K,P)
    ),
    ( ? resistor(s,F,L,J)
    ; ? resistor(s,F,J,L)
    ),
    ( ? resistor(s,G,L,N)
    ; ? resistor(s,G,N,L)
    ),
    ( ? resistor(s,H,N,K)
    ; ? resistor(s,H,K,N)
    ),
    ? circuito(s,Q,regulador_de_tensao,A,entrada(K,L),
     saida(O),potencia(K,L),conexoes(K,L,O)),
    restricoes([N,O,M,P],[K,L]).
```

* **Terceira situação:**

O *netlist* apresentado é:

```
TRANSISTOR COMO DIODO
* TERMINAIS = 1,2
Q1 2 2 1 M
.END
```

1)Bloco de Entrada

A forma interna obtida é:

```
tran_BIPOC5,q1,2,2,1)
```

2)Bloco de generalização construtiva 1

Não existe uma regra na base de regras que possa ser uma componente não terminal para esta estrutura exemplo.

3)Bloco de generalização seletiva

A regra obtida tem a forma:

```
circuito(5,3,diodo,[diodo,A],entrada,saida,potencia,
conexoes(B,C)) :-
    P tran_BIPOC5,A,C,C,B),
    restricoes([],[]).
```

Esta forma da regra corresponde a forma final a ser acrescentada na base de regras.

4)Bloco de generalização construtiva 2

Esta última regra pode ser utilizada como uma subcomponente para a primeira regra armazenada na base de regras.

O acréscimo desta última regra e a modificação da primeira regra da base de regras, com base nesta última regra, tornam o conteúdo da base de regras da forma:

BASE DE REGRAS

```
circuito(s,2,comparador,[comparador,A,B,C,D,E,F,G,H],
entrada(I),saida(J),potencia(K,L),conexoes(K,L,I,J)):-
    ? tran_BIPO(s,B,J,I,M),
    ? tran_BIPO(s,C,L,N,M),
    ? tran_BIPO(s,D,M,O,P),
    ( ? resistor(s,E,P,K)
    ; ? resistor(s,E,K,P)),
    ( ? resistor(s,F,L,J)
    ; ? resistor(s,F,J,L)),
    ( ? resistor(s,G,L,N)
    ; ? resistor(s,G,N,L)),
    ( ? resistor(s,H,N,K)
    ; ? resistor(s,H,K,N)),
    ? circuito(s,Q,regulador_de_tensao,A,entrada(K,L),
saida(O),potencia(K,L),conexoes(K,L,O)),
    restricoes([N,O,M,P],[K,L]).
circuito(s,1,regulador_de_tensao,[regulador_de_tensao,A,B,C],
entrada(D,E),saida(F),potencia(D,E),conexoes(D,E,F)):-
    ( ? resistor(s,B,E,F)
    ; ? resistor(s,B,F,E)),
    ? circuito(s,G,diodo,C,entrada,saida,potencia,
conexoes(H,F)),
    ? circuito(s,I,diodo,A,entrada,saida,potencia,
conexoes(D,H)),
    restricoes([H],[D,E]).
circuito(s,3,diodo,[diodo,A],entrada,saida,potencia,
conexoes(B,C)) :-
    ? tran_BIPO(s,A,C,C,B),
    restricoes([],[]).
```

Estas três situações mostradas ilustram as possibilidades que ocorrem durante a criação de regras a partir dos exemplos. A primeira situação ocorre somente uma vez durante a inicialização de uma nova base de regras. As situações 2 e 3

podem ocorrer isoladas ou simultaneamente.

O apêndice A mostra um conjunto de circuitos da família lógica TTL [16,19], e o resultado do acréscimo das regras correspondentes a estes circuitos na base de regras.

4.2 - Classificação de circuitos

O processo de classificação dos circuitos exemplo, assemelha-se ao processo de análise sintática de frases. As regras armazenadas na base de regras são utilizadas no reconhecimento de estruturas no circuito. A seguir estão alguns exemplos de aplicação das regras no processo de classificação.

A listagem das regras encontra-se no apêndice A, bem como os circuitos que deram origem a elas.

Netlist apresentada

```
COMPARADOR
*IN = 9
* OUT = 10
* POT = 1,2
Q1 10 9 11 M
Q2 2 3 11 M
Q3 11 4 12 M
Q4 4 4 5 M
Q5 5 5 1 M
R1 12 1 M
R2 2 10 M
R3 2 3 M
R4 3 1 M
R5 2 4 M
.END
```

Classificação

```
comparador:
  regulador_de_tensão:
    diodo:
      q5
    r5
    diodo:
      q4
q1
q2
q3
r1
r2
r3
r4
entrada: 9
saida: 10
potencia: 1 2
```

```
BUFFER 7407
* ENTRADA 2
* SAIDA 8
Q1 4 3 2 M
Q2 5 4 6 M
Q3 7 6 0 M
Q4 8 7 0 M
D1 0 2 M
R1 1 3 M
R2 1 5 M
R3 6 0 M
R4 1 7 M
.END
```

```
buffer_7407:
  inversor_7405:
    r3
    r2
    r1
    d1
    q3
    q2
    q1
  q4
  r4
entrada: 2
saida: 8
potencia: 0 1
```

```
PORTA NOU N7402
*ENTRADAS 2 3
*SAIDA 12
Q1 5 4 2 M
Q2 6 5 7 M
Q3 6 8 7 M
Q4 8 9 3 M
Q5 10 6 11 M
Q6 12 7 0 M
D1 0 2 M
D2 0 3 M
D3 11 12 M
R1 1 4 M
R2 1 6 M
R3 7 0 M
R4 1 9 M
R5 1 10 M
.END
```

```
porta_NOU_7402:
  saida_transistor_diodo:
    r5
    d3
    q6
    q5
  q1
  q2
  q3
  q4
  d1
  d2
  r1
  r2
  r3
  r4
entrada: 2 3
saida: 12
potencia: 0 1
```

O processo de classificação pode ser dirigido para a busca de estruturas específicas. Abaixo está o resultado da busca por um circuito *saida_transistor_diodo* pertencente a última netlist acima (7402):

```
saida_transistor_diodo:
  r5
  d3
  q6
  q5
entrada: 6 7
saida: 12
potencia: 0 1
```

A implementação das regras em linguagem PROLOG torna possível a obtenção de respostas alternativas para o processo de classificação. Abaixo está uma resposta alternativa correspondente a *netlist* de uma porta lógica do CI7402:

```
Classificação:
porta_NOU_7402:
  saida_transistor_diodo:
    r5
    d3
    q6
    q5
  q4
  q3
  q2
  q1
  d2
  d1
  r4
  r2
  r3
  r1
```

entrada: 3 2
saida: 12
potencia: 0 1

Esta resposta revela uma simetria em relação aos terminais de entrada de uma porta lógica do CI7402.

4.3 - Considerações

Alguns pontos importantes podem ser observados quando da criação de regras pelo sistema. O sistema é capaz de criar regras e armazená-las de maneira compacta, utilizando no seu corpo elementos não terminais. Este processo de criação de regras é, porém, limitado. Para executar esta tarefa a estrutura da regra candidata a ser incluída como uma chamada de um não terminal é procurada dentro da estrutura exemplo, e se for encontrada, será substituída por um termo PROLOG indicando a existência desta estrutura. Este processo é repetido até que dentro da estrutura exemplo não haja mais nenhuma estrutura cuja regra seja conhecida. Disto resulta que a sequência de apresentação dos exemplos para criação da regra é importante e define a forma final das regras na base de regras. Duas sequências de exemplos formadas pelos mesmos componentes, porém em ordens diferentes de apresentação, não resultam necessariamente no mesmo conjunto de regras final. Isto ocorre porque no processo de busca de estruturas conhecidas no exemplo, pode haver uma intersecção de estruturas correspondentes a regras diferentes, e a escolha de uma das regras elimina a possibilidade das outras regras formarem um elemento não terminal da regra sendo criada. A alteração da sequência de exemplos altera o conjunto de regras disponíveis à cada exemplo criado e assim modifica o conjunto de regras final obtido. Esta situação de disputa de regras para formação de um não terminal do corpo de uma nova regra, é resultado da opção de implementação que objetiva a obtenção de somente uma regra a partir de cada exemplo

fornecido. A criação de todas as regras possíveis para cada exemplo promove um aumento exagerado de regras na base de regras podendo tornar o sistema inviável em termos de ocupação de memória, tempo de execução e gerenciamento do conteúdo da base de regras, pois a cada regra criada deve ser feita uma pesquisa entre todas as regras na base de regras, visando a execução do segundo processo de generalização construtiva, além da execução de processos secundários.

Portanto, durante a criação de um conjunto de regras, deve ser observada a ordem de apresentação dos exemplos, visando a obtenção de regras adequadas.

Apesar do sistema contar com uma opção de retiradas de regras já assimiladas, cuidados devem ser tomados na sua utilização. Deve ser observado que, devido as consequências internas na base de regras, provocadas pela assimilação de uma determinada regra, devem ser evitados os processos de retirada de regras. O estado resultante da base de regras com a retirada de uma regra pode não ser o mesmo que seria obtido se a regra a ser retirada nunca tivesse sido criada.

CAPÍTULO 5

CONCLUSÕES

O sistema desenvolvido consiste num ambiente de armazenamento otimizado de regras de reconhecimento de estruturas de circuito eletrônicos, as quais são criadas a partir da apresentação de exemplos. Este sistema constitui uma aplicação de princípios de aprendizagem a partir de exemplos seguindo técnicas de otimização de armazenamento de conhecimento [35,36] e técnicas de manipulação de regras e descrições [1,20]. Como substrato para aplicação destas técnicas utilizou-se cláusulas da linguagem PROLOG. As descrições dos circuitos constituídas por *netlist's* são traduzidas para cláusulas PROLOG e são posteriormente tratadas pelo sistema com a finalidade de obtenção de regras de reconhecimento. A *netlist* constitui uma linguagem artificial para descrição de circuitos que facilita o processo de aprendizagem, pois as informações nela contidas são sempre relevantes a descrição do circuito e podem ser usadas na sua identificação.

O conjunto de regras obtido possui características adequadas de portabilidade e legibilidade herdadas da linguagem PROLOG. A portabilidade advém da criação das regras de maneira que constituam um conjunto de cláusulas PROLOG que descrevem um procedimento de reconhecimento independente. O sistema visa a utilização destas regras de maneira eficiente, através de mecanismos de busca. A legibilidade das regras é consequência direta da linguagem em que estão implementadas. Esta legibilidade permite o pronto entendimento das regras, e torna possível a atuação direta do usuário sobre elas. As descrições topológicas dos circuitos eletrônicos são facilmente expressas pelas cláusulas PROLOG, e devido a generalidade desta notação, informações não topológicas podem ser acrescidas às regras de reconhecimento sem alterar a homogeneidade da forma da regra.

A opção pelo enunciado das regras em PROLOG traz como consequência uma maior modularidade e legibilidade das regras obtidas; porém, outras opções poderiam ser escolhidas, como por

exemplo o mecanismo de rede semântica [22,33]. As redes semânticas constituem um mecanismo eficiente de armazenamento de conhecimento. Através da sua utilização o mecanismo de escolha de somente uma regra por exemplo apresentado poderia ser evitado, e consequentemente cada exemplo poderia atuar de maneira mais eficiente para criação de modelos de estruturas dos circuitos eletrônicos. Em contrapartida, este corpo de conhecimento não seria tão legivel, ou portátil, quanto o sistema apresentado.

A produção de somente uma regra por exemplo constitui um subaproveitamento das informações nele contidas, e a utilização do mecanismo de redes semânticas, possibilita o total aproveitamento desta informação. O aproveitamento total da informação pode não ser uma opção adequada, pois engloba o armazenamento de informações, que podem não ser totalmente relevantes no processo de classificação da estrutura de circuitos eletrônicos. A adoção de medidas heurísticas pode servir como um meio para tornar o sistema de regras menos ineficiente na absorção de informações, possibilitando que seja criado um conjunto de regras mais adequado a partir de cada exemplo fornecido. No desenvolvimento do sistema foi dada ênfase na topologia dos circuitos, de maneira que o reconhecimento de estruturas leva em consideração somente este aspecto da descrição do circuito, ignorando os valores dos componentes eletrônicos. Esta atitude pode ser inicialmente adotada porém limita a capacidade de reconhecimento funcional de uma estrutura. Uma mesma função pode ser realizada por meio de várias estruturas diferentes (isomorfismo funcional), o que ocorre muito frequentemente com funções lógicas, porém, o inverso também ocorre. Uma mesma estrutura pode ter sua função alterada, dependendo dos valores do seus componentes ou dos sinais elétricos incidindo sobre eles, como é o caso de conjuntos de portas lógicas que executam funções determinadas por sinais de controle. O reconhecimento deste segundo tipo de função não pode ser feito somente pela topologia do circuito. é necessário que, além disto, o sistema identifique as consequências de um determinado sinal elétrico sobre o estado elétrico da topologia,

e para isso é necessário um conhecimento quantitativo dos componentes do circuito.

SUGESTÕES

O sistema proposto constitui um protótipo que pode ser melhorado de maneira a atuar em áreas específicas. Como aplicação em reconhecimento topológico, além da classificação de circuitos podemos reconhecer erros estruturais de projetos em circuitos eletrônicos, desde que sejam apresentadas informações para formação das regras correspondentes. O sistema pode ser associado a um capturador de esquemático, de maneira à auxiliar a análise e projeto de circuitos eletrônicos.

As regras podem ser incrementadas, de maneira que possam utilizar os valores e características de cada elemento básico do circuito eletrônico, como uma medida para o discernimento entre a classificação dos circuitos e determinação de sua função. O número dos elementos básicos permitidos pelas regras pode ser aumentado.

Além do reconhecimento topológico, as regras podem ser aperfeiçoadas no sentido de capturar a influência dos estados elétricos da topologia na determinação de sua função.

As regras podem ser melhoradas no sentido de descreverem de maneira mais compacta as regularidades iterativas das estruturas dos circuitos.

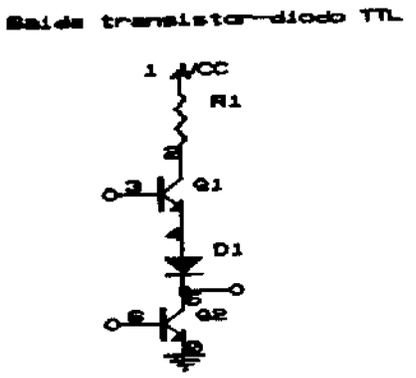
O mecanismo de classificação funciona como uma interpretação estrutural para o *netlist* apresentado. O inverso pode ser imaginado, onde as regras podem ser usadas no modo gerativo. Desta forma dada uma especificação, seria possível a geração da *netlist* correspondente. Um maior detalhamento das informações manipuláveis pela regra, permitiriam a geração de *netlist*'s mais específicas. Ainda imaginando o uso das regras nesta direção, podemos incrementar as regras de maneira que tratem o *netlist* à nível de implementação em CI (circuito integrado). Desta maneira a especificação de um circuito, pode

resultar no desenvolvimento automático do *layout* que o implemente. Versões mais avançadas das regras poderiam ser associadas à sistemas de explicação automática do funcionamento de circuitos e sistemas de diagnósticos.

NETLIST's e ESQUEMÁTICOS

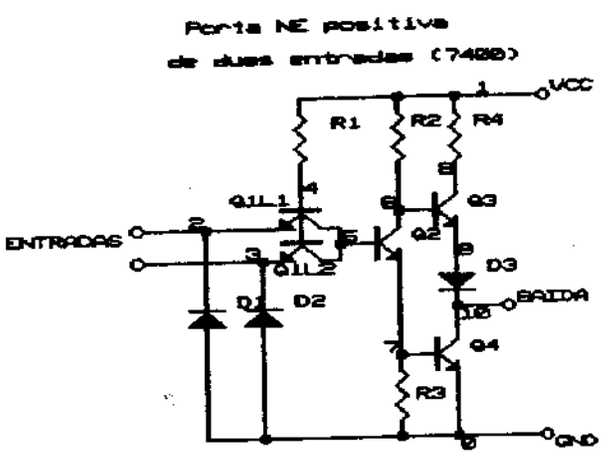
CONFIGURACAO DE SAIDA TTL [16]: TRANSISTOR-DIODO

```
*ENTRADAS 3,6
*SAIDA 5
Q1 2 3 4 M
Q2 5 6 0 M
D1 4 5 M
R1 1 2 M
.END
```

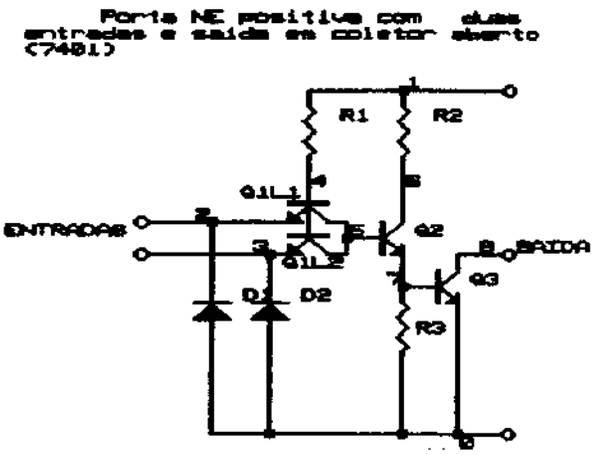


PORTAS LOGICAS TTL [19]: UMA PORTA NE (7400)

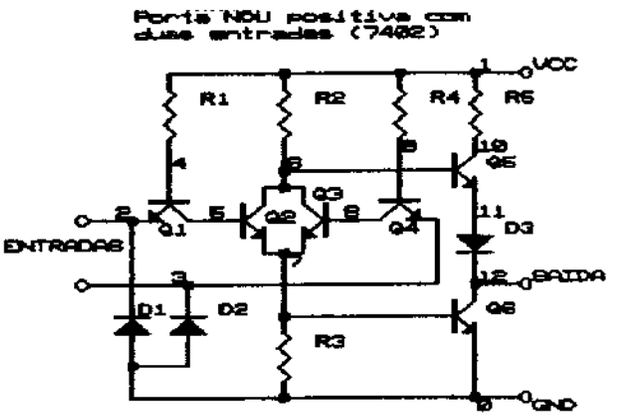
```
*ENTRADAS 2 3
*SAIDA 10
Q1L1 5 4 2 M
Q1L2 5 4 3 M
Q2 6 5 7 M
Q3 8 6 9 M
Q4 10 7 0 M
D1 0 2 M
D2 0 3 M
D3 9 10 M
R1 1 4 M
R2 1 6 M
R3 7 0 M
R4 1 8 M
.END
```



PORTAS LOGICAS TTL [193]: UMA PORTA NE (7401) COM SAIDA EM
*COLETOR ABERTO
*ENTRADAS 2 3
* SAIDA 8
Q1L1 5 4 2 M
Q1L2 5 4 3 M
Q3 6 5 7 M
Q4 8 7 0 M
D1 0 2 M
D2 0 3 M
R1 1 4 M
R2 1 6 M
R3 7 0 M
.END

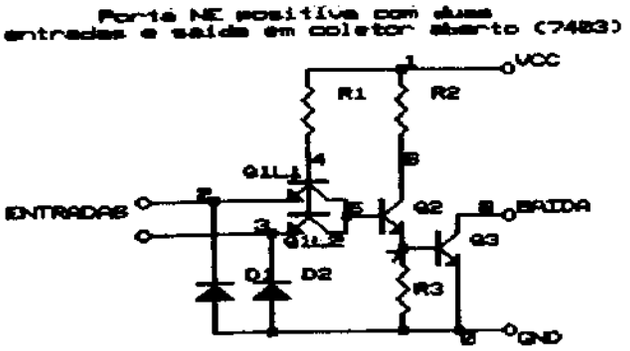


PORTAS LOGICAS TTL[193]: PORTA NOU POSITIVA (7402)
*ENTRADAS 2 3
*SAIDA 12
Q1 5 4 2 M
Q2 6 5 7 M
Q3 6 8 7 M
Q4 8 9 3 M
Q5 10 6 11 M
Q6 12 7 0 M
D1 0 2 M
D2 0 3 M
D3 11 12 M
R1 1 4 M
R2 1 6 M
R3 7 0 M
R4 1 9 M
R5 1 10 M
.END



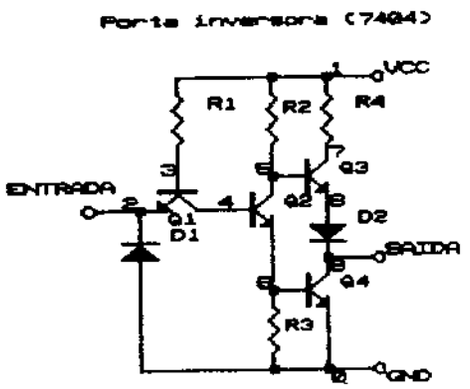
PORTAS LOGICAS TTL [19]: PORTA POSITIVA NE (7403)

```
*ENTRADAS 2 3
*SAIDA 8
Q1L1 5 4 2 M
Q1L2 5 4 3 M
Q2 6 5 7 M
Q3 8 7 0 M
D1 0 2 M
D2 0 3 M
R1 1 4 M
R2 1 6 M
R3 7 0 M
.END
```

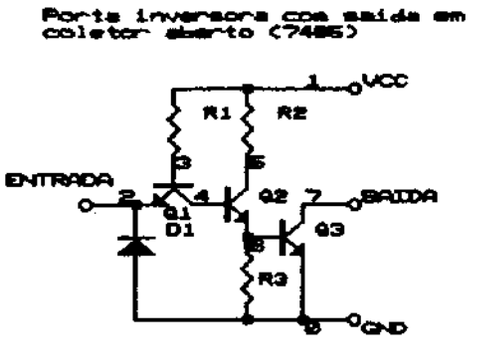


PORTAS LOGICAS TTL [19]:PORTA INVERSORA (7404)

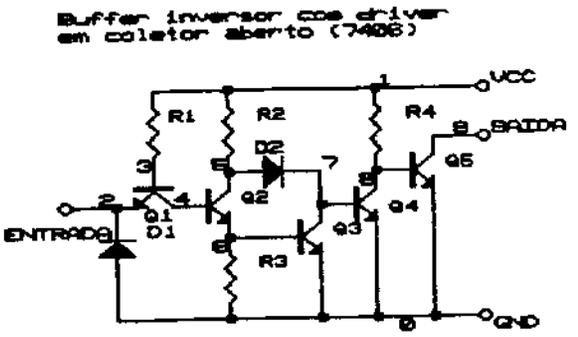
```
*ENTRADA 2
*SAIDA 9
Q1 4 3 2 M
Q2 5 4 6 M
Q3 7 5 8 M
Q4 9 6 0 M
D1 0 2 M
D2 8 9 M
R1 1 3 M
R2 1 5 M
R3 6 0 M
R4 1 7 M
.END
```



PORTAS LOGICAS TTL [19]:PORTA INVERSORA COM SAIDA EM
*COLETOR ABERTO (7405)
*ENTRADA 2
*SAIDA 7
Q1 4 3 2 M
Q2 5 4 6 M
Q3 7 6 0 M
D1 0 2 M
R1 1 3 M
R2 1 5 M
R3 6 0 M
.END

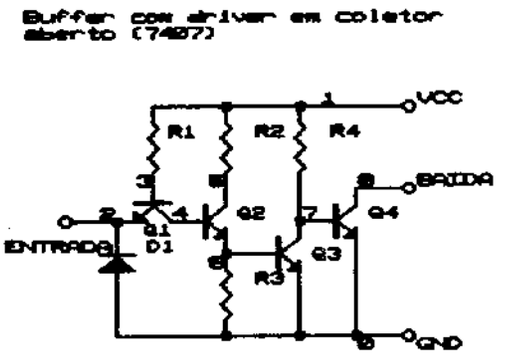


PORTAS LOGICAS TTL [19]: BUFFER INVERSOR, DRIVER COM COLETOR
*ABERTO (7406)
* IN = 2
* OUT = 9
Q1 4 3 2 M
Q2 5 4 6 M
Q3 7 6 0 M
Q4 8 7 0 M
Q5 9 8 0 M
D1 0 2 M
D2 5 7 M
R1 1 3 M
R2 1 5 M
R3 6 0 M
R4 1 8 M
.END



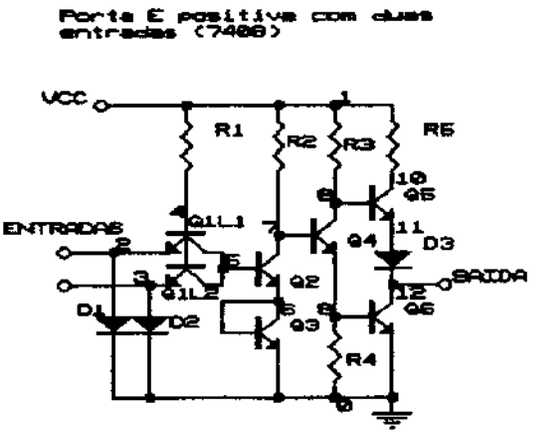
TTL[19]: BUFFER COM DRIVER EM COLETOR ABERTO (7407)

```
* ENTRADA 2
* SAIDA 8
Q1 4 3 2 M
Q2 5 4 6 M
Q3 7 6 0 M
Q4 8 7 0 M
D1 0 2 M
R1 1 3 M
R2 1 5 M
R3 6 0 M
R4 1 7 M
.END
```



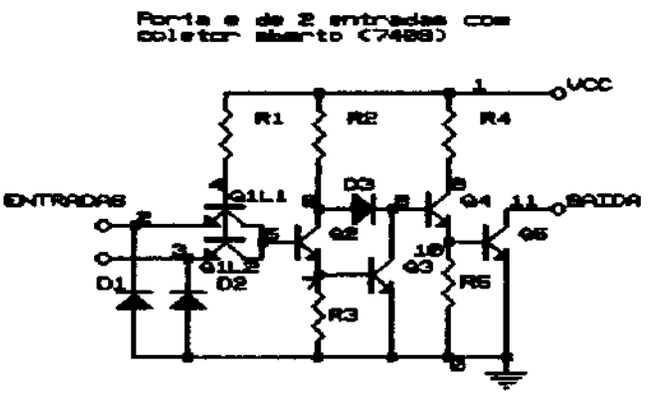
PORTAS LOGICAS TTL[19]: PORTA E POSITIVA (7408)

```
* ENTRADAS 2,3
* SAIDAS 12
Q1L1 5 4 2 M
Q1L2 5 4 3 M
Q2 7 5 6 M
Q3 6 6 0 M
Q4 8 7 9 M
Q5 10 8 11 M
Q6 12 9 0 M
D1 0 2 M
D2 0 3 M
D3 11 12 M
R1 1 4 M
R2 1 7 M
R3 1 8 M
R4 9 0 M
R5 1 10 M
.END
```



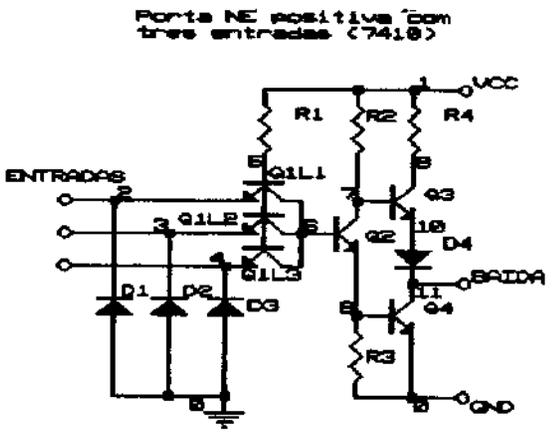
PORTAS LOGICAS TTL[19]:PORTA E COM SAIDA EM COLETOR ABERTO (7409)

```
* ENTRADA 2,3
* SAIDA 11
Q1L1 5 4 2 M
Q1L2 5 4 3 M
Q2 6 5 7 M
Q3 8 7 0 M
Q4 9 8 10 M
Q5 11 10 0 M
D1 0 2 M
D2 0 3 M
D3 6 8 M
R1 1 4 M
R2 1 6 M
R3 7 0 M
R4 1 9 M
R5 10 0 M
.END
```



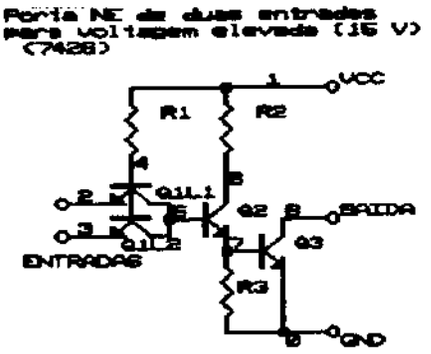
PORTAS LOGICAS TTL[19]: PORTA NE POSITIVA COM 3 ENTRADAS (7410)

```
*ENTRADAS 2,3,4
*SAIDA 11
Q1L1 6 5 2 M
Q1L2 6 5 3 M
Q1L3 6 5 4 M
Q2 7 6 8 M
Q3 9 7 10 M
Q4 11 8 0 M
D1 0 2 M
D2 0 3 M
D3 0 4 M
D4 10 11 M
R1 1 5 M
R2 1 7 M
R3 8 0 M
R4 1 9 M
.END
```



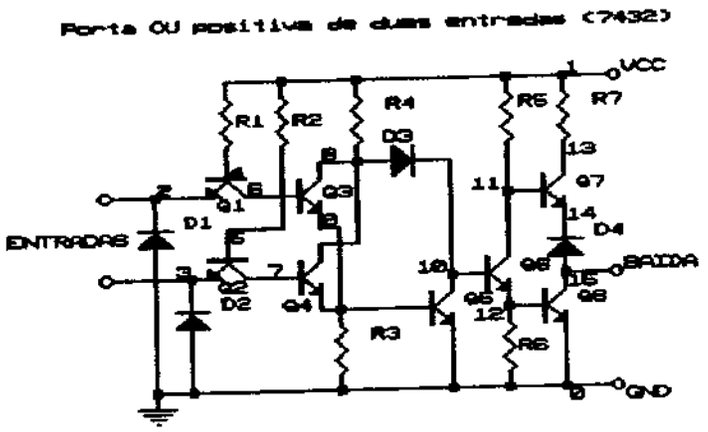
PORTAS LOGICAS TTL[19]: PORTA NE PARA VOLTAGEM ELEVADA (15 V)

```
* (7426)
* ENTRADA 2,3
* SAIDA 8
Q1L1 5 4 2 M
Q1L2 5 4 3 M
Q2 6 5 7 M
Q3 8 7 0 M
R1 1 4 M
R2 1 6 M
R3 7 0 M
.END
```



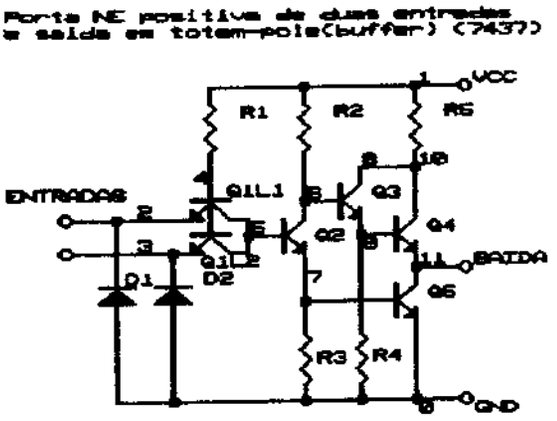
PORTAS LOGICAS TTL[19]: PORTA OU POSITIVA (7432)

```
* ENTRADA 2 3
* SAIDA 15
Q1 6 4 2 M
Q2 7 5 3 M
Q3 8 6 9 M
Q4 8 7 9 M
Q5 10 9 0 M
D1 0 2 M
D2 0 3 M
D3 8 10 M
D4 15 14 M
R1 1 4 M
R2 1 5 M
R3 9 0 M
R4 1 8 M
R5 1 11 M
R6 12 0 M
R7 1 13 M
.END
```



PORTAS LOGICAS TTL[19]: PORTA NE POSITIVA (7437)

```
* ENTRADA 2,3
* SAIDA 11
Q1L1 5 4 2 M
Q1L2 5 4 3 M
Q2 6 5 7 M
Q3 8 6 9 M
Q4 10 9 11 M
Q5 11 7 0 M
D1 0 2 M
D2 0 3 M
R1 1 4 M
R2 1 6 M
R3 7 0 M
R4 9 0 M
.END
```



CONJUNTO DE REGRAS RESULTANTES NA BASE DE REGRAS

```
circuito(s,2,comparador,[comparador,A,B,C,D,E,F,G,H],entrada(I),
saida(J),potencia(K,L),conexoes(K,L,I,J)):-
    ? tran_BIPO(s,B,J,I,M),
    ? tran_BIPO(s,C,L,N,M),
    ? tran_BIPO(s,D,M,O,P),
    ( ? resistor(s,E,P,K)
    ; ? resistor(s,E,K,P)
    ),
    ( ? resistor(s,F,L,J)
    ; ? resistor(s,F,J,L)
    ),
    ( ? resistor(s,G,L,N)
    ; ? resistor(s,G,N,L)
    ),
    ( ? resistor(s,H,N,K)
    ; ? resistor(s,H,K,N)
    ),
    ? circuito(s,Q,regulador_de_tensao,A,entrada(K,L),saida(O),
potencia(K,L),conexoes(K,L,O)),
    restricoes([N,O,M,P],[K,L]).

circuito(s,1,regulador_de_tensao,[regulador_de_tensao,A,B,C],
entrada(D,E),saida(F),potencia(D,E),conexoes(D,E,F)):-
    ( ? resistor(s,B,E,F)
    ; ? resistor(s,B,F,E)
    ),
    ? circuito(s,G,diodo,C,entrada,saida,potencia,conexoes(H,F)),
    ? circuito(s,I,diodo,A,entrada,saida,potencia,conexoes(D,H)),
    restricoes([H],[D,E]).

circuito(s,3,diodo,[diodo,A],entrada,saida,potencia,conexoes(B,C)) :-
    ? tran_BIPO(s,A,C,C,B),
    restricoes([],[]).
```

```
circuito(s,4,saida_transistor_diodo,[saida_transistor_diodo,A,B,C,D],
entrada(E,F),saida(G),potencia(H,I),conexoes(H,I,E,G,F)):-
    ? tran_BIPO(s,C,G,F,H),
    ? tran_BIPO(s,D,J,E,K),
    ? diodo(s,B,K,G),
    ( ? resistor(s,A,I,J)
    ; ? resistor(s,A,J,I)
    ),
    restricoes([J,K],[H,I]).

circuito(s,5,porta_NE_7400,[porta_NE_7400,A,B,C,D,E,F,G,H,I],
entrada(J,K),saida(L),potencia(M,N),conexoes(M,N,J,K,L)):-
    ? tran_BIPO(s,B,O,P,J),
    ? tran_BIPO(s,C,O,P,K),
    ? tran_BIPO(s,D,Q,O,R),
    ? diodo(s,E,M,J),
    ? diodo(s,F,M,K),
    ( ? resistor(s,G,N,P)
    ; ? resistor(s,G,P,N)
    ),
    ( ? resistor(s,H,N,Q)
    ; ? resistor(s,H,Q,N)
    ),
    ( ? resistor(s,I,R,M)
    ; ? resistor(s,I,M,R)
    ),
    ? circuito(s,S,saida_transistor_diodo,A,entrada(Q,R),
saida(L),potencia(M,N),conexoes(M,N,Q,L,R)),
    restricoes([P,O,Q,R],[M,N]).
```

```
circuito(s,7,porta_NOU_7402,[porta_NOU_7402,A,B,C,D,E,F,G,H,I,J,K],
entrada(L,M),saida(N),potencia(O,P),conexoes(O,P,L,M,N)),
    ? tran_BIPO(s,B,Q,R,L),
    ? tran_BIPO(s,C,S,Q,T),
    ? tran_BIPO(s,D,S,U,T),
    ? tran_BIPO(s,E,U,V,M),
    ? diodo(s,F,O,L),
    ? diodo(s,G,O,M),
    ( ? resistor(s,H,P,R)
    ; ? resistor(s,H,R,P)
    ),
    ( ? resistor(s,I,P,S)
    ; ? resistor(s,I,S,P)
    ),
    ( ? resistor(s,J,T,O)
    ; ? resistor(s,J,O,T)
    ),
    ( ? resistor(s,K,P,V)
    ; ? resistor(s,K,V,P)
    ),
    ? circuito(s,W,saida_transistor_diodo,A,entrada(S,T),saida(N),
potencia(O,P),conexoes(O,P,S,N,T)),
    restricoes([R,Q,S,T,U,V],[O,P]).
```

```
circuito(s,8,inversor_7404,[inversor_7404,A,B,C,D,E,F,G],entrada(H),
saida(I),potencia(J,K),conexoes(J,K,H,I)) :-
    ? tran_BIPO(s,B,L,M,H),
    ? tran_BIPO(s,C,N,L,O),
    ? diodo(s,D,J,H),
    ( ? resistor(s,E,K,M)
    ; ? resistor(s,E,M,K)
    ),
    ( ? resistor(s,F,K,N)
    ; ? resistor(s,F,N,K)
    ),
    ( ? resistor(s,G,O,J)
    ; ? resistor(s,G,J,O)
    ),
    ? circuito(s,P,saida_transistor_diodo,A,entrada(N,O),saida(I),
potencia(J,K),conexoes(J,K,N,I,O)),
    restricoes([M,L,N,O],[J,K]).

circuito(s,9,inversor_7405,[inversor_7405,A,B,C,D,E,F,G],entrada(H),
saida(I),potencia(J,K),conexoes(J,K,H,I)):-
    ? tran_BIPO(s,E,I,L,J),
    ? tran_BIPO(s,F,M,N,L),
    ? tran_BIPO(s,G,N,O,H),
    ? diodo(s,D,J,H),
    ( ? resistor(s,A,L,J)
    ; ? resistor(s,A,J,L)
    ),
    ( ? resistor(s,B,K,M)
    ; ? resistor(s,B,M,K)
    ),
    ( ? resistor(s,C,K,O)
    ; ? resistor(s,C,O,K)
    ),
    restricoes([O,N,M,L],[J,K]).
```

```
circuito(s,10,buffer_inversor_7406,
[buffer_inversor_7406,A,B,C,D,E,F,G,H,I,J,K],entrada(L),saida(M),
potencia(N,O),conexoes(N,O,L,M)) :-
    ? tran_BIPO(s,G,M,P,N),
    ? tran_BIPO(s,H,P,Q,N),
    ? tran_BIPO(s,I,Q,R,N),
    ? tran_BIPO(s,J,S,T,R),
    ? tran_BIPO(s,K,T,U,L),
    ? diodo(s,E,S,Q),
    ? diodo(s,F,N,L),
    ( ? resistor(s,A,O,P)
    ; ? resistor(s,A,P,O)
    ),
    ( ? resistor(s,B,R,N)
    ; ? resistor(s,B,N,R)
    ),
    ( ? resistor(s,C,O,S)
    ; ? resistor(s,C,S,O)
    ),
    ( ? resistor(s,D,O,U)
    ; ? resistor(s,D,U,O)
    ),
    restricoes([U,T,S,R,Q,P],[N,O]).

circuito(s,11,buffer_7407,[buffer_7407,A,B,C],entrada(D),saida(E),
potencia(F,G),conexoes(F,G,D,E)) :-
    ? tran_BIPO(s,B,E,H,F),
    ( ? resistor(s,C,G,H)
    ; ? resistor(s,C,H,G)
    ),
    ? circuito(s,I,inversor_7405,A,entrada(D),saida(H),
potencia(F,G),conexoes(F,G,D,H)),
    restricoes([H],[F,G]).
```

```
circuito(s,12,porta_E_7408,[porta_E_7408,A,B,C,D,E,F,G,H,I,J,K,L],
entrada(M,N),saida(O),potencia(P,Q),conexoes(P,Q,M,N,O)):-
    ? tran_BIPO(s,C,R,S,M),
    ? tran_BIPO(s,D,R,S,N),
    ? tran_BIPO(s,E,T,R,U),
    ? tran_BIPO(s,F,V,T,W),
    ? diodo(s,G,P,M),
    ? diodo(s,H,P,N),
    ( ? resistor(s,I,Q,S)
    ; ? resistor(s,I,S,Q)
    ),
    ( ? resistor(s,J,Q,T)
    ; ? resistor(s,J,T,Q)
    ),
    ( ? resistor(s,K,Q,V)
    ; ? resistor(s,K,V,Q)
    ),
    ( ? resistor(s,L,W,P)
    ; ? resistor(s,L,P,W)
    ),
    ? circuito(s,X,diodo,A,entrada,saida,potencia,conexoes(P,U)),
    ? circuito(s,Y,saida_transistor_diodo,B,entrada(V,W),saida(O),
potencia(P,Q),conexoes(P,Q,V,O,W)),
    restricoes([S,R,U,T,V,W],[P,Q]).
```

```
circuito(s,13,porta_E_7409,
[porta_E_7409,A,B,C,D,E,F,G,H,I,J,K,L,M,N],entrada(O,P),saida(Q),
potencia(R,S),conexoes(R,S,O,P,Q)):-
    ? tran_BIPO(s,I,Q,T,R),
    ? tran_BIPO(s,J,U,V,T),
    ? tran_BIPO(s,K,V,W,R),
    ? tran_BIPO(s,L,X,Y,W),
    ? tran_BIPO(s,M,Y,Z,P),
    ? tran_BIPO(s,N,Y,Z,O),
    ? diodo(s,F,X,V),
    ? diodo(s,G,R,P),
    ? diodo(s,H,R,O),
    ( ? resistor(s,A,T,R)
    ; ? resistor(s,A,R,T)
    ),
    ( ? resistor(s,B,S,U)
    ; ? resistor(s,B,U,S)
    ),
    ( ? resistor(s,C,W,R)
    ; ? resistor(s,C,R,W)
    ),
    ( ? resistor(s,D,S,X)
    ; ? resistor(s,D,X,S)
    ),
    ( ? resistor(s,E,S,Z)
    ; ? resistor(s,E,Z,S)
    ),
    restricoes([Z,Y,X,W,V,U,T],[R,S]).
```

```
circuito(s,14,porta_NE_7410,[porta_NE_7410,A,B,C,D,E,F,G,H,I,J,K],
entrada(L,M,N),saida(O),potencia(P,Q),conexoes(P,Q,L,M,N,O)):-
    ? tran_BIPO(s,B,R,S,L),
    ? tran_BIPO(s,C,R,S,M),
    ? tran_BIPO(s,D,R,S,N),
    ? tran_BIPO(s,E,T,R,U),
    ? diodo(s,F,P,L),
    ? diodo(s,G,P,M),
    ? diodo(s,H,P,N),
    ( ? resistor(s,I,Q,S)
    ; ? resistor(s,I,S,Q)
    ),
    ( ? resistor(s,J,Q,T)
    ; ? resistor(s,J,T,Q)
    ),
    ( ? resistor(s,K,U,P)
    ; ? resistor(s,K,P,U)
    ),
    ? circuito(s,V,saida_transistor_diodo,A,entrada(T,U),saida(O),
potencia(P,Q),conexoes(P,Q,T,O,U)),
    restricoes([S,R,T,U],[P,Q]).

circuito(s,6,porta_NE_7401,[porta_NE_7401,A,B,C],entrada(D,E),
saida(F),potencia(G,H),conexoes(G,H,D,E,F)):-
    ? diodo(s,C,G,D),
    ? diodo(s,B,G,E),
    ? circuito(s,I,porta_NE_7426,A,entrada(E,D),saida(F),
potencia(G,H),conexoes(G,H,E,D,F)),
    restricoes([],[G,H]).
```

```
circuito(s,15,porta_NE_7426,[porta_NE_7426,A,B,C,D,E,F,G],
entrada(H,I),saida(J),potencia(K,L),conexoes(K,L,H,I,J)):-
    ? tran_BIPO(s,D,J,M,K),
    ? tran_BIPO(s,E,N,O,M),
    ? tran_BIPO(s,F,O,P,I),
    ? tran_BIPO(s,G,O,P,H),
    ( ? resistor(s,A,M,K)
    ; ? resistor(s,A,K,M)
    ),
    ( ? resistor(s,B,L,N)
    ; ? resistor(s,B,N,L)
    ),
    ( ? resistor(s,C,L,P)
    ; ? resistor(s,C,P,L)
    ),
    restricoes([P,O,N,M],[K,L]).
```

```
circuito(s,16,porta_OU_7432,
[porta_OU_7432,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P],entrada(Q,R),
saida(S),potencia(T,U),conexoes(T,U,Q,R,S)):-
    ? tran_BIPO(s,L,V,W,T),
    ? tran_BIPO(s,M,X,Y,W),
    ? tran_BIPO(s,N,X,Z,W),
    ? tran_BIPO(s,O,Y,AA,R),
    ? tran_BIPO(s,P,Z,AB,Q),
    ? diodo(s,H,S,AC),
    ? diodo(s,I,X,V),
    ? diodo(s,J,T,R),
    ? diodo(s,K,T,Q),
    ( ? resistor(s,A,U,AD)
    ; ? resistor(s,A,AD,U)
    ),
    ( ? resistor(s,B,AE,T)
    ; ? resistor(s,B,T,AE)
    ),
    ( ? resistor(s,C,U,AF)
    ; ? resistor(s,C,AF,U)
    ),
    ( ? resistor(s,D,U,X)
    ; ? resistor(s,D,X,U)
    ),
    ( ? resistor(s,E,W,T)
    ; ? resistor(s,E,T,W)
    ),
    ( ? resistor(s,F,U,AA)
    ; ? resistor(s,F,AA,U)
    ),
    ( ? resistor(s,G,U,AB)
    ; ? resistor(s,G,AB,U)
    ),
    restricoes([AB,AA,Z,Y,X,W,V,AF,AE,AD,AC],[T,U]).
```

```
circuito(s,17,porta_NE_7437,
[porta_NE_7437,A,B,C,D,E,F,G,H,I,J,K,L],entrada(M,N),saida(O),
potencia(P,Q),conexoes(P,Q,M,N,O)):-
     ? tran_BIPO(s,G,O,R,P),
     ? tran_BIPO(s,H,S,T,O),
     ? tran_BIPO(s,I,U,V,T),
     ? tran_BIPO(s,J,V,W,R),
     ? tran_BIPO(s,K,W,X,N),
     ? tran_BIPO(s,L,W,X,M),
     ? diodo(s,E,P,N),
     ? diodo(s,F,P,M),
     ( ? resistor(s,A,T,P)
     ; ? resistor(s,A,P,T)
     ),
     ( ? resistor(s,B,R,P)
     ; ? resistor(s,B,P,R)
     ),
     ( ? resistor(s,C,Q,V)
     ; ? resistor(s,C,V,Q)
     ),
     ( ? resistor(s,D,Q,X)
     ; ? resistor(s,D,X,Q)
     ),
     restricoes([X,W,V,R,U,T,S],[P,Q]).
```

BIBLIOGRAFIA

[1] Bundy, A.; Silver, B. e Plummer, D., "An Analytical Comparison of Some Rule-Lerning Programs", *Artificial Intelligence* 27 (1985), pag. 137-181.

[2] Butchelor, Bruce G., *Pattern Recognition - Ideas in Practice*, Plenum Press-New York and London, 1978.

[3] Casanova, M. A. et al, *Programação em Lógica e a Linguagem Prolog*, Ed. Edgar Blücher Ltda, 1987.

[4] Church, Alonzo, *Introduction to Mathematical Logic*, Princeton University Press, 1956.

[5] Dietterich,T.G. e Michalski,R.S., "Learning and Generalization of Characteristics Descriptions: Evaluation Criteria and Comparative Review of Selected Methods", 6ª *IJCAI* (1979), pag. 223-231.

[6] Ellman, T. - "Generalizing logic circuit designs by analyzing proofs of correctness", 9ª *IJCAI* (1985), pag. 644-646.

[7] Fu, K. S., *Syntatic Pattern Recognition and Applications*, Prentice Hall, 1982.

[8] Fu, K. S. e Booth, T. L., "Grammatical inference Introduction and Survey - Part I", *IEEE Transactionns on Systems, Man and Cybernetics*, Vol. SMC-5, número 1, janeiro de 1975, pag. 95-111.

[9] Fu, K. S. e Booth, T. L., "Grammatical inference Introduction and Survey - Part II", *IEEE Transactionns on Systems, Man and Cybernetics*, Vol. SMC-5, número 4, julho de 1975, pag. 409-423.

[10] Fu, K. S. e Brayer, J. M., "Some Multidimensional grammar inference methods", *Pattern Recognition and Artificial Intelligence*, Chen, C. H. (ed.), Academic Press Inc, 1976.

[11] Gonzalez, R.C. e Thomason, M. G., *Syntatic Pattern Recognition - An Introduction*, Addison-Wesley Publishing Company, 1978.

[12] Hayes-Roth, F., "Patterns of induction and Associated Knowledge acquisition algorithms", *Pattern Recognition and Artificial Intelligence*, Chen, C. H (ed.), Academic Press Inc, 1976.

[13] Hayes-Roth, F., "Knowledge acquisition from structural descriptions", 5º *IJCAI* (1977), pag. 356-362.

[14] Hayes-Roth, F. e McDermott, J., "An Interference Matching Technique for Inducing Abstractions", CACM, vol. 21, número 5, maio 1978, pag. 401-411.

[15] Hedrick, C.L., "Learning Production Systems from Examples", *Artificial Intelligence 7* (1976), pag.21-49.

[16] Kawakami, P., *Signetics Logic TTL DATA MANUAL*, 1978.

[17] Arity Corporation, *The Arity/Prolog Language Reference Manual*, 1988.

[18] MicroSim Corporation, *PSPICE User's Guide*, 1984.

[19] Signetics, *TTL Logic Data Manual*, 1982.

[20] Michalki, R.S., Carbonell, J.G. e Mitchell, T.M., *Machine Learning : An Artificial Intelligence Approach*, Morgan Kaufmann Publishers Inc., 1983.

[21] Michalski, R. S., "Pattern Recognition as Rule-Guided Inductive Inference", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-2, número4, julho 1980, pag. 349-361.

[22] Nilsson, Nils J., *Principles of Artificial Intelligence*, Springer Verlag, 1982.

[23] Pavlidis, T., *Structural Pattern Recognition*, Spring-Verlag, 1977.

[24] Pereira, Fernando C. N. e Warren, David H. D., "Definite Clause Grammars for Language Analysis - A survey of the formalism and a Comparison with Augmented Transition Networks", *Artificial Intelligence* 13(1980), 231-278.

[25] Quinlan, J.R., "Learning Efficient Classification Procedures and their Applications to Chess end games", *Machine Learning : An Artificial Intelligence Approach*, Eds. Michalski, R.S., Carbonel, G. e Mitchell, T., Kaufmann Publishers Inc., 1983.

[26] Quinlan, J.R., "Semi-autonomous acquisition of pattern-based knowledge", *Machine Intelligence* 10, Halsted Press, 1982, pag. 159-172.

[27] Sowa, J.F., *Conceptual Structures Information Processing in Mind and Machine*, Addison-Wesley Publishing Company, 1984.

[28] Tanaka, Takushi, " Parsing Circuit Topology in a deductive system ", 9ª *IJCAI* (1985), pag.407-410.

[29] Vere, A. S., "Relational Production Systems", *Artificial Intelligence* 8 (1977), pag. 47-68.

[30] Vere, A. S., "Induction of Relational Productions in the Presence of Background Information", 5ª *IJCAI* (1977), pag. 349-355.

[31] Vere, A. S., "Inductive Learning of Relational Productions",em *Pattern-Directed Inference Systems*, Watterman, Hayes-Roth, Academic Press 1978, pag. 281-295.

[32] Vere, A. S., "Multilevel Counterfactuals for Generalizations of Relational Concepts and Productions", *Artificial Intelligence* ,número 14 (1980), pag. 139-164.

[33] Winston, Patrick Henry, *Artificial Intelligence*, Addison Wesley Publishing Co.

[34] Winston, P.H., "Learning structural descriptions from examples", em *The Psychology of computer vision*, Winston, P. H., McGraw-Hill Book Company.

[35] Wolff, J.G., "Language acquisition and the discovery of phrase struture", *Language and Speech* , vol. 23, número 3, 1980, pag. 255-269.

[36] Wolff, J. G., "Cognitive Development as Optimisation", *Symbolic Computation*, Leonard Bolc (Ed.), *Computacional Models of Learning*, Springer Verlag.