Uso de Linguagem Natural para Programação de Controladores Lógico-Programáveis

Guilherme Nelson Fernandes de Souza

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

Aprovada por:

Profª. Sueli Bandeira Teixeira Mendes(Ph.D.)
(Presidente)

Profª. Leila Maria Ripoll Eizirik (D.Sc.)

Prof José Lucas Mourão Rangel Netto(Ph.D.)

RIO DE JANEIRO, RJ - BRASIL
ABRIL DE 1991

SOUZA, GUILHERME NELSON FERNANDES DE

Uso de Linguagem Natural para Programação de Controladores Lógico-Programáveis (Rio de Janeiro) 1991.

X, 102p. 29.7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1991)

Tese - Universidade Federal do Rio de Janeiro, COPPE.

1. Linguagem Natural

2. Controladores Lógico-Programáveis

I. COPPE/UFRJ II. Título (Série)

A Luiza,

Nelson, Thiago e Thomaz

Agradecimentos

A minha Nininha e a meus filhos, por tudo que abriram mão de fazer.
A meus pais, por tudo que fizeram e também que não fizeram.
A Sueli, pelo muito que me ajudou e fez.
Aos amigos*, do Cepel, da COPPE e do dia-a-dia, por serem cúmplices do que fiz.
Ao Cepel, por tornar factível.
Mas principalmente a Deus, sem quem, nada, ninguém teria feito.

---

* Em especial a Homero Gonçalves de Andrade, Carlos Alberto Ferreira e Márcio Oliveira Rocha. Ao Mauricio Moszkowicz pela amizade e apoio.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para obtenção do grau de Mestre em Ciências (M.Sc.).

USO DE LINGUAGEM NATURAL PARA PROGRAMAÇÃO DE CONTROLADORES LÓGICO-PROGRAMÁVEIS

GUILHERME NELSON FERNANDES DE SOUZA

ABRIL DE 1991

ORIENTADOR: Ph.D. Sueli B. T. Mendes

PROGRAMA: Programa de Engenharia de Sistemas e Computação

Para a implementação da Lógica de Controle de uma Usina Hidrelétrica são utilizadas Linguagens Artificiais ou formalismos complexos que levam à introdução de erro no processo de descrição desta Lógica.

Este trabalho sugere a utilização de um Sistema Inteligente e Interativo com interface em Linguagem Natural para descrição do Controle, como solução para este problema.

São apresentadas algumas das técnicas mais comuns no Processamento de Linguagem Natural, até ser proposto um novo formalismo chamado RVG-LC (Register Vector Grammar - Livre de Contexto) que se baseia no modelo RVG, de G. Blank, dando a este um poder de representação maior sem aumento consideravel de complexidade.

É também feita uma comparação entre RVG-LC e ATN, baseando-se em dois fenômenos gramaticais facilmente resolvidos por este último formalismo.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.).

USE OF NATURAL LANGUAGE FOR PROGRAMMING OF PROGRAMMABLE LOGIC CONTROLERS

GUILHERME NELSON FERNANDES DE SOUZA

APRIL OF 1991

SUPERVISOR: Ph.D. Sueli B. T. Mendes

DEPARTMENT: Programa de Engenharia de Sistemas e Computação

This Thesis describes the problems faced when implementing the Control Logic of a Power Plant using conventional Artificial Languages.

It suggests the use of Natural Language in a Interactive and Intelligent System for Configuration of the Control as an solution which reduces error generation during the description process.

A presentation of the most used techinics on Natural Language Processing precedes the proposal of a new one called RVG-CF (Register Vector Grammar - Context-Free). This formalism capture the original philosophy behind Blank's model RVG, putting more representation power on it with little complexity increase.

RVG-CF is compared with ATN within tipical gramatical phenomena well solved in ATN formalism.

ÍNDICE DE FIGURAS E TABELAS

CAPÍTULO I

INTRODUÇÃO

Muito se tem feito no intuito de prover interfaces "amigáveis" a sistemas computacionais. A evidência mais presente deste esforço hoje, está nas interfaces GUI (Graphics User Interface) largamente utilizadas em ambientes operacionais (Windows para DOS, X11 e Motif para Unix), planilhas de cálculo (Excel, Lotus), gerenciadores de bancos de dados e muitos outros aplicativos e utilitários.

No entanto, está cada vez mais patente a necessidade de tornar o uso do computador uma tarefa mais simples. A solução já está sendo esboçada no desenvolvimento de interfaces por voz - reconhecimento e síntese de voz - por alguns grupos e centros de pesquisa.

É dentro desta ótica, desenvolvimento de interfaces amigáveis, que esta tese se posiciona, procurando demonstrar que um sistema interativo com interface em Linguagem Natural é tão eficiente e conciso para se descrever a lógica de controle de uma Usina Hidrelétrica quanto um sistema convencional, que utiliza Linguagens Artificiais como Ladder, Lógica Combinacional ou MS (Ling. para Máquinas Sequenciais - [MIRAND 89]). Apresentando, ainda, a vantagem de ser universalmente compreendido.

1.1. HISTÓRICO

O aumento da demanda de energia que vem ocorrendo nas últimas décadas no país despertou os governos no sentido de planejar um programa de expansão do sistema elétrico. Para isso, está prevista a construção de diversas usinas hidrelétricas até o ano 2010.

Apesar dos índices de nacionalização de uma usina já serem bastante altos, no que diz respeito à instrumentação o país ainda importa todos os equipamentos

[ELETRO 87]. Dentro deste contexto o Cepel, juntamente com a Eletrosul e Furnas, vem desenvolvendo tecnologia própria para supervisão e controle de usinas e subestações, objetivando a transferência desta para a indústria nacional.

Como parte desta tecnologia, foi desenvolvido o Sequenciador de Partida, Parada e Operação de Unidades Geradoras. Este equipamento é um controlador lógico-programável no qual se encerram todos os procedimentos necessários para, conforme o nome indica, partir, parar, e operar as unidades geradoras de uma usina hidrelétrica [MIRAND 89].

Os requisitos de confiabilidade, disponibilidade e segurança envolvidos no projeto do Sequenciador exigiram um estudo profundo das arquiteturas possíveis de hardware e software, até se chegar à solução atual.

Deste estudo, no que diz respeito ao software, saíram algumas conclusões e orientações a serem seguidas. A principal orientação foi no sentido de se dividir o software em duas partes: básico e aplicativo. O objetivo era reunir as partes comuns no software básico e assim, com as diversas versões e instalações diferentes, conseguir um estágio de maturação elevado (quanto a erros nos algoritmos ou nas implementações). No software aplicativo, então, estariam as partes específicas de cada usina. Este software, além de não ser exaustivamente testado, seria projetado e implementado por especialistas no processo e não em programação.

Essas considerações conduziram à definição de uma metodologia de descrição e implementação da lógica de controle, contida no software aplicativo, com as seguintes características: clareza e concisão do modelo de representação da lógica de controle, simplicidade de implementação, mantendo-se o mais próximo possível da modelagem.

Tradicionalmente, os controladores lógico-programáveis utilizam descrições em diagramas Ladder ou lógicos. Apesar de usarem lógica combinacional, estes diagramas permitem a representação de elementos de memorização, mas de uma

forma não explícita. Após uma série de experiências com diversos modelos, sempre buscando uma boa aceitação por parte dos projetistas, chegou-se a um modelo de máquina de estados com autômatos em estrutura estratificada [MIRAND 89].

Foi então criada uma linguagem (MS - Máquinas Sequenciais) para representar as máquinas de estados projetadas. Esta linguagem pode representar, de uma maneira bem simples, todas as entidades de uma máquina de Mealy e as suas interligações.

A solução atual representa um grande avanço tecnológico em relação aos métodos anteriores, que exigiam do especialista, além do conhecimento do processo, habilidade para traduzir tal conhecimento para uma lógica de relés relativamente complexa e, por consequência, sujeita a erros.

Deve-se frisar que o objetivo de um modelo de máquina de estado e do uso da linguagem MS era aproximar a implementação da lógica de controle do seu modelo, ou seja, da descrição desta lógica em máquinas de Mealy [BOOTH 67]. Com isso evita-se a introdução de erro durante a implementação.

1.2. PROGRAMAÇÃO EM LINGUAGEM NATURAL

Durante o desenvolvimento de qualquer tecnologia existe sempre uma tendência a se perder de vista o objetivo básico que estimulou a sua introdução e, depois dos primeiros sucessos, o interesse tende, naturalmente, a se concentrar na tecnologia em si e nos problemas que ela apresenta.

No caso da tecnologia de computadores essa regra não é diferente. Ao longo do tempo, nos esquecemos que o objetivo original não era apenas desenvolver sistemas cada vez mais poderosos e rápidos, mas aumentar a eficiência dos computadores em problemas de decisão ou solução tipicamente exclusivos ao homem [LEDGAR 80].

Dentro desta visão, o desenvolvimento de interfaces utilizando processamento de Linguagem Natural resgata, ainda, outro objetivo, que é o de tornar o uso dos computadores uma tarefa menos árdua e universalmente acessível.

Segundo [SIQUEI 88], para que um maior número de pessoas tenha acesso à informação contida num computador, digitando perguntas em português, uma série de questões precisa ser respondida:

a) qual registro da Língua Portuguesa o computador tem, atualmente, condições de aprender?
b) como ensinar uma Língua ao computador? De que maneira ele irá aprender?
c) uma vez que apenas uma parte da Língua Portuguesa pode ser ensinada, como devemos treinar o usuário nesta Linguagem Natural Restrita? O uso de vocabulário aparente aumenta a eficiência na interação usuário-computador?
d) como tratar o problema da "explosão" do modelo e, principalmente, da implementação quando do aumento do dicionário ou das construções gramaticais?
e) qual o ganho, quantitativa e qualitativamente, na adoção de Linguagens Naturais Restritas versus Linguagens Artificiais?

Nos sistemas já desenvolvidos, a LN que tem sido possível ensinar ao computador é uma Linguagem Natural Restrita.

Uma Linguagem Natural Restrita é um registro da LN em que são eliminadas possíveis redundâncias e excluídas muitas opções, isto é, uma Linguagem que contém os elementos imprescindíveis para que uma comunicação, referente a um campo específico de conhecimento, se estabeleça via teclado-vídeo.

Nesta definição informal de Linguagem Natural Restrita pode estar contida a resposta para uma das questões propostas. Um "campo específico" do conhecimento pode implicar no uso de jargão e no uso de um pequeno sub-conjunto da Gramática Portuguesa, o que impediria a explosão da implementação e eliminaria a necessidade de Vocabulário Aparente ou treinamento do usuário no Sistema de Linguagem Natural Restrita.

Mais de 40% dos sistemas utilizando Processamento de Linguagem Natural de que se tem referência são interfaces a banco de dados [OBERME 87] e as consultas que propiciam são do tipo:

Quais os funcionários que ganham mais de Cr$50.000,00 de salário por mês?

Quais os funcionários cujas férias estão vencidas e que ganham mais de Cr$30.000,00 de salário por mês?

Liste o turno de trabalho, os anos de casa, as horas extras e a data de aquisição de férias dos funcionários do departamento comercial.

Estas sentenças[1] estão bem distantes das que usamos no dia-a-dia, para nos comunicarmos com pessoas cujo repertório é resultante de vivências extremamente diferenciadas.

[1] Será adotada a palavra 'sentença' para designar o que [CUNHA 72] e [SACCON 87] classificam como período

"- Maria, ponha isso lá fora em qualquer parte.

- Junto com as outras?

- Não, senão pode vir alguém e querer fazer qualquer coisa com elas. Ponha no lugar do outro dia.

- Sim, senhora. Olha, o homem está aí.

- Aquele de quando choveu?

- Não, o que a senhora foi lá e falou com ele no domingo."

Millôr Fernandes

E mais distante ainda da Linguagem presente nos textos literários.

"Acabado o artefazer é que a obra-de-arte principia vivendo."

Mario de Andrade

1.3. ORIENTAÇÃO DO TRABALHO

Este trabalho de Tese foi orientado a atacar três frentes de estudo. A primeira destas frentes ou linhas de atuação constou de uma pesquisa bibliográfica bastante intensa da qual se buscou retirar justificativas para a adoção de Interfaces em Linguagem Natural para sistemas computacionais em geral. Estas justificativas serviram para responder a questão sobre as vantagens em se adotar Linguagem Natural diante de Linguagem Artificial, dando uma maior segurança e certeza na escolha.

A segunda linha de atuação objetivava resolver o problema do setor elétrico, qual seja: programar com segurança o Sistema de Controle Digital (Controlador Lógico-Programável) para executar a Lógica de Controle de uma Usina

Hidrelétrica. Para tanto, uma série de entrevistas com o especialista no processo elétrico foi realizada na busca de se conhecer e modelar o sistema elétrico.

Como resultado destas duas frentes, surgiu a proposta de se criar um Ambiente ou Sistema de Desenvolvimento para a Programação da Lógica de Controle. Tal Sistema é dotado de Interface em Linguagem Natural e é, ao mesmo tempo, justificado e motivado pelo estudo realizado.

Na última das frentes buscou-se estudar o Processamento de Linguagem Natural, as técnicas utilizadas e os problemas que se apresentam. O resultado é a proposta de uma nova técnica de PLN que foi batizada como RVG-LC, por ser baseada em outra já existente, RVG. A técnica proposta aumenta esta última de maneira a gozar de características apreciadas em formalismos mais difundidos como Redes Recursivas Aumentadas (ATN), Gramáticas Livres de Contexto, Gramáticas Frase Estruturadas, Lógica com Cláusulas Definidas (DCG) etc.

Os capítulos que se seguem estão divididos em três partes. A primeira destas partes consiste dos capítulos 2, 3 e 4 e traduzem as duas primeiras frentes de estudo, ou seja, é onde se pretende justificar e propôr o uso de Linguagem Natural. Para tanto são apresentadas (cap. 2) algumas conclusões que foram tiradas de uma seleção de trabalhos publicados sobre o assunto. O intuito é demonstrar as vantagens na aplicação de Processamento de Linguagem Natural em sistemas computacionais de uma maneira geral.

O capítulo 3 apresenta o problema específico do setor elétrico e tem por objetivo chamar a atenção da dificuldade em se utilizar Linguagens de programação convencionais, ou qualquer outro modelo formal, na programação de sistemas de controle. Sugere-se, assim, que o uso de Linguagem Natural é uma solução razoável.

O último capítulo desta primeira parte apresenta a solução proposta; um ambiente interativo para descrição da lógica de controle utilizando Linguagem Natural.

A segunda parte trata dos conceitos e técnicas para o Processamento de Linguagem Natural e consiste dos capítulos 5 e 6.

No capítulo 5 são apresentadas as duas técnicas mais utilizadas em PLN: ATN e DCG.

RVG, uma técnica recentemente proposta, é detalhadamente explicada e comparada com as demais. Uma vez compreendido o modelo, são sugeridas algumas modificações de forma a torna-lo mais poderoso do ponto de vista da capacidade de representação da gramática da LN (RVG-LC). São propostas, também, duas metodologias aplicaveis na utilização de RVG: uma para conversão da notação Frase-Estruturada para RVG e outra de ATN para RVG.

A terceira e última parte resume os trabalhos de implementação da solução proposta. Algoritmos, representações da Lingua Portuguesa em RVG, entre outros resultados que foram produzidos são apresentados ao longo do capítulo 7 e apêndices.

CAPÍTULO II

LINGUAGEM NATURAL

X

LINGUAGEM ARTIFICIAL

Algoritmos descritos em Linguagem Natural são fragmentos de sentenças independentes umas das outras, um conjunto fracamente organizado. Tais descrições são normalmente muito robustas, graças ao grau de redundância que apresentam e frequentemente concisas dado o conhecimento inato do leitor e seu domínio da aplicação [SCHWAR 74].

A questão mais importante, antes de iniciar a especificação e implementação de um sistema para descrição dos algoritmos de controle em Linguagem Natural, é quanto às vantagens frente a um sistema utilizando uma Linguagem Artificial já que com esta pode-se obter descrições igualmente concisas e robustas.

Existem alguns artigos que mostram as vantagens no uso de Linguagem Natural em sistemas computacionais.

Abaixo estão resumidos alguns considerados mais categóricos quanto a essa vantagem.

O primeiro trabalho, de Idméa Siqueira, "Aspectos Psico-Linguísticos envolvidos na interação Homem-Máquina em um Sistema de Linguagem Natural", aborda o lado psico-social do uso de LNR em sistemas de interface HM.

Neste trabalho, várias pessoas com níveis de treinamento e escolaridade diferentes foram observadas durante a interação com um sistema de consulta a banco de dados com interface em LN, concluindo-se :

"O sistema é bem aceito, principalmente pelos usuários menos treinados."

"O SLN se tornará mais eficaz quanto mais o usuário estiver sensibilizado para utilizar a LNR, ou seja, quando o modelo do interlocutor[2] for o modelo realista."

"A existência de Vocabulário Aparente no SLN aciona estratégias cognitivas do usuário oferecendo pistas que facilitarão o uso do modelo realista de interlocutor."

Outro trabalho, de Robert F. Simmons, "Man-Machine Interfaces: Can they guess what you want?", faz um levantamento sobre no que um sistema em LN pode contribuir para a melhoria dos sistemas computacionais, além da facilidade de interação. As conclusões são :

"Com o crescimento na capacidade e complexidade dos Sistemas Computacionais, se tornou necessário provê-los com várias informações e sistemas de 'Help'. (...) 'Mouses' e sistemas de menu são, atualmente, os melhores avanços em se tratando de interfaces amigáveis.(...) Os Sistemas atuais não incluem modelos explícitos dos usuários ou de suas intenções, mas são projetados com a filosofia de prover apenas as informações que os usuários necessitam para operar o sistema com sucesso. Já os SLN devem ir mais além, eles devem não só guiar o usuário na sua aplicação, mas também prever e fornecer informações que o usuário quer, mesmo que as questões não especifiquem a sua intenção."[3]

Exemplo :

P. - Existe algum trem para Zurich?

R. - Sim. Plataforma 5, às 21:00h.

ou

R. - Não, mas um ônibus expresso deixa a estação diariamente às 24:00h.

---

[2] modelo do interlocutor = modelo que o usuário cria para representar o sistema em LNR

[3] esta forma implícita de informação é conhecida como ato de fala indireta - indirect speech act - [ALLEN 80] [SEARLE 75]

Nestes dois casos o sistema pressupõe[4] que o usuário não quer saber apenas da existência de um trem para Zurich, mas os meios de que ele dispõe para sair da estação e detalhes sobre tais meios.

Um contra exemplo seria um sistema que não faz nenhuma pressuposição.

P. - Quais os alunos reprovados em COS801 em 1986?

R. - Nenhum.

P. - Alguém tirou grau C em COS801 em 1986?

R. - Não.

P. - Quais os alunos que foram aprovados com grau A ou B?

R. - Nenhum.

P. - Foi oferecida COS801 em 1986?

R. - Não.

Em "The Natural Language of Interactive Systems", de Henry Ledgard et alii, é feita uma medida do aumento da eficiência expressa pelo número de tarefas concluídas com e sem erro, percentagem de comandos errados e do volume final de trabalho alcançado com LNxLA.

Concluindo :

"O resultado demonstra que o editor com comandos em inglês alcançou melhor desempenho."

"Em todas as medidas, o desempenho usando o editor em inglês foi superior ao do mesmo usando linguagem notacional. Isto foi verdade não importando o nível de experiência dos usuários."

---

[4] mais sobre pressuposição em LN em [KEENAN], [GARNER], [LANGEN], [FILLMO]

Por último, mas não menos importante, H. Albert Napier et alii em "Impact of a RNL Interface on Ease of Learning and Productivity" mostram o ganho em se adotar LNR para usuários que não estejam acostumados com qualquer linguagem artificial ou sistema computacional.

Conclusões :

"Os resultados demonstram uma clara e consistente vantagem em favor do Lotus HAL[5]."

"Vantagens significativas em relação ao Lotus 123 foram obtidas tanto em aceitação como performance."

Este último trabalho aborda o caso exato no qual nos enquadramos. Usuários especializados no processo elétrico têm que especificar a lógica de controle da unidade geradora em MS (Linguagem Artificial) quando poderiam fazê-lo em Linguagem Natural.

A conclusão geral a que podemos chegar é que um sistema em Linguagem Natural é mais eficiente que em Linguagem Artificial, para especialistas e principalmente para os não especialistas em sistemas computacionais.

Esta eficiência pode estar relacionada a várias explicações:

a) por aceitar melhor um SLN o usuário não se predispõe a errar,
b) o usuário está "melhor treinado" em LN do que LA, ou seja, está acostumado a traduzir seu pensamento em LN,
c) a estrutura mais livre da LN, apesar de gerar ambiguidades, é mais rica na semântica, indicando a intenção principal do usuário e várias outras implícitas. Todas numa única sentença.

[5] com interface em Linguagem Natural

CAPÍTULO III

DESCRIÇÃO DO PROBLEMA

Neste capítulo é apresentada a organização do Sistema Elétrico no Brasil. São rapidamente descritos todos os componentes do sistema, desde o nível mais alto na hierarquia até os elementos de uma unidade geradora: relés, disjuntores, etc. No final é discutida a forma de atuação sobre tais elementos, o que constitui a lógica de controle propriamente dita.

3.1. SISTEMA INTERLIGADO

A energia elétrica no Brasil tem sua geração e transmissão feitas de forma interligada. Isto significa que as diversas concessionárias de energia, por todo o país, estão conectadas numa única rede de geração e transmissão. A interrupção na geração de uma usina é superada pelo aumento imediato na capacidade de geração de uma outra usina, em qualquer parte da rede.

Cada empresa concessionária de energia possui um sistema elétrico próprio, composto de usinas, termelétricas ou hidrelétricas, subestações e linhas de transmissão. Os sistemas das diversas concessionárias se interligam através dos pontos de intercâmbio de energia.

Do ponto de vista operacional, o sistema interligado possui uma estrutura hierárquica que pode ser vista na Figura 3.1 e é composto pelos seguintes Centros:

a) Centro Nacional de Supervisão e Coordenação (CNSC) - está sob a responsabilidade da Eletrobrás. No topo da hierarquia, coordena a operação dos COS's através do Sistema Interligado de Supervisão e Coordenação (SINSC).

b) Centro de Operação do Sistema (COS) - está sob a responsabilidade de cada concessionária de energia. Tem por objetivo o controle e a supervisão do seu sistema elétrico próprio, além do controle no intercâmbio de energia com as demais concessionárias.

c) Centro de Operação Regional (COR) - responsável pela operação de usinas e subestações. Realiza, entre outras funções, o controle das grandezas como tensão, corrente, etc, a coordenação das manobras e trabalhos de manutenção e a normalização após perturbações. As funções do COR podem, dependendo da área coberta pela concessionária, ser realizadas no COS.

d) Centro de Operação de Usina ou Subestação - (COU/COSU) - respondem pelo controle direto sobre uma usina ou subestação.

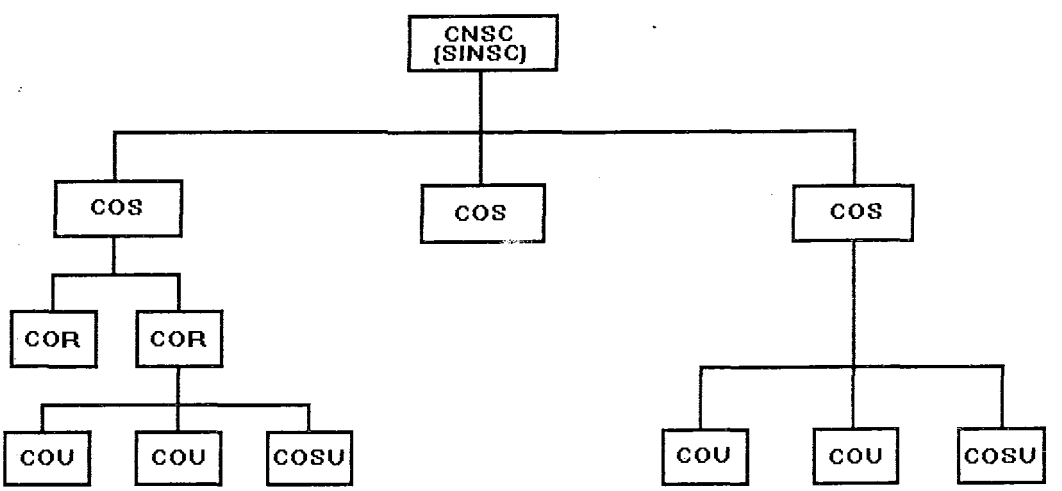


FIGURA 3.1 - HIERARQUIA DO SISTEMA INTERLIGADO

3.2. USINA HIDRELÉTRICA

Uma usina hidrelétrica é responsável por gerar energia elétrica a partir da energia cinética da água, captada no final de uma queda. É composta de uma ou várias unidades geradoras.

Uma unidade geradora da usina pode ser dividida em três subsistemas:

a) Subsistema de Geração de Energia - é composto pelos equipamentos responsáveis pela geração de energia como: turbina, gerador, excitatriz, mancais e etc.

b) Subsistema de Conexão ao Sistema Elétrico - é composto pelos elementos de conexão à rede elétrica; disjuntores, chaves seccionadoras, barras e etc.

c) Subsistema de Controle de Vazão Efluente - é responsável pelo controle da vazão de água por toda a unidade geradora. É composto de comportas, válvulas e condutos.

Em linhas gerais o funcionamento de uma unidade geradora é: a água é captada e forçada pelo conduto até chegar a turbina. O movimento da água faz a turbina girar. Este movimento é transmitido ao rotor do gerador que está acoplado ao mesmo eixo da turbina. No gerador está, também, o estator, ambos formados por placas de enrolamentos interligadas. Um outro gerador bem menor, chamado de ímã permanente (PMG - Permanent Magnetic Generator) gera uma corrente elétrica que produz um campo magnético na excitatriz. Este campo, por sua vez, produz uma outra corrente elétrica, através dos enrolamentos do rotor, que irá induzir a tensão final de saída da unidade geradora para o sistema elétrico. Esta tensão é elevada por um transformador a valores padronizados.

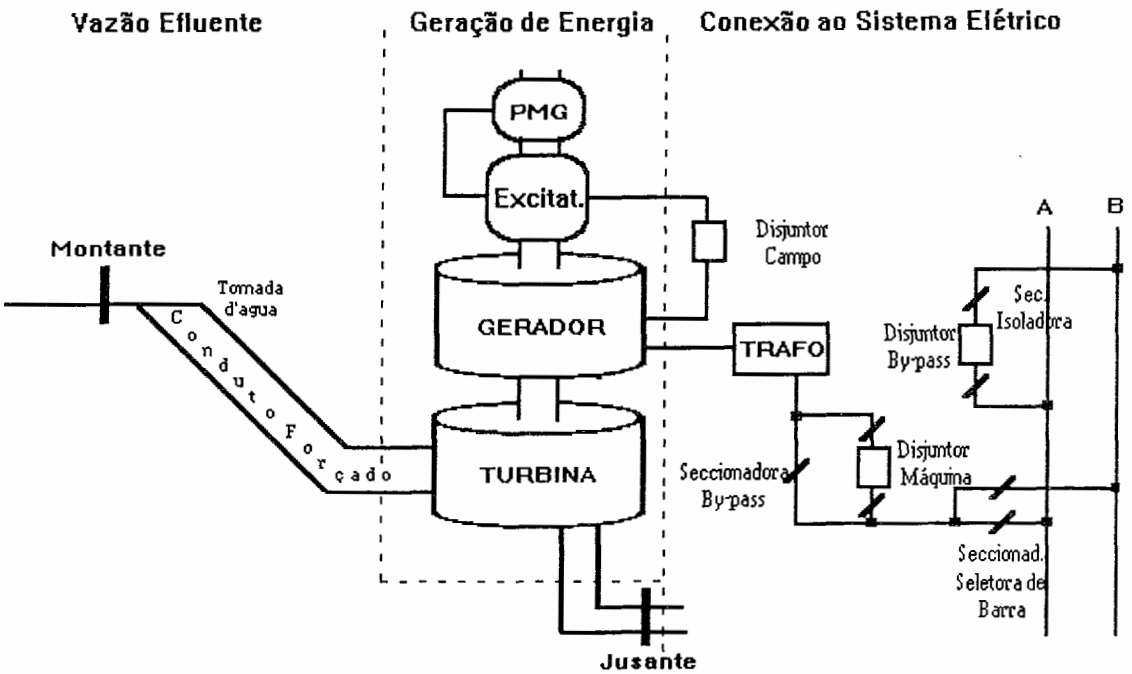


FIGURA 3.2 - ESQUEMA DE UMA UNIDADE GERADORA

A conexão com o sistema elétrico é então feita via disjuntores e chaves seccionadoras que se ligam às barras principal(A) e secundária(B) onde se conecta todo o sistema interligado.

Para realizar esta sua função básica a unidade dispõe dos seguintes componentes:

Disjuntores - São chaves para grande potência e acionamento rápido (10 ms)

Campo - interrompe a corrente aplicada ao rotor e que gera o campo magnético do gerador

Máquina - conecta o transformador a uma das barras (principal ou secundária)

By-pass - conecta a barra principal à secundária. Este disjuntor substitui o de máquina quando do isolamento do mesmo para manutenção ou verificação preventiva

Seccionadoras - São chaves para acionamento sem carga, ou seja sem corrente. Têm acionamento lento (1 min)

Isoladoras - isolam os disjuntores do resto do circuito elétrico para manutenção ou verificação preventiva

By-pass - substituem o disjuntor que for isolado

Seletoras de Barra - selecionam a qual barra o disjuntor de máquina, ou seccionadora de By-pass correspondente, estará ligado, ou seja, determina em qual barra estará ligada a unidade geradora

Válvulas - Controlam a passagem de água ou ar no subsistema de vazão efluente

Esférica - válvula de acionamento bastante lento (1 a 3 min) e serve para interromper a entrada de água na turbina

Solenóide de Abertura do Distribuidor - controla a abertura do distribuidor, permitindo uma gradação da quantidade de água admitida na turbina. Com isso controla-se a quantidade de movimento que se aplicará ao rotor

Solenóide de Rebaixamento do Tubo de Sucção - quando do fechamento da válvula esférica e da necessidade de se expulsar a água que inunda a turbina, esta válvula é aberta injetando-se ar no tubo de sucção

Des-aeração do Tubo de Sucção - controla a retirada de ar-comprimido do tubo de sucção

Borboleta - interrompe a passagem de água no conduto forçado

Bomba de Cunha de Óleo - injeta óleo nos mancais de sustentação das partes girantes da turbina

Compressor de Ar - injeta ar nos tanques de rebaixamento

Reguladores - Sistemas de malha fechada para controle de algumas grandezas da unidade

Tensão - controla a tensão gerada. Atua sobre o sistema de excitação que determina a quantidade de campo magnético que será aplicada ao estator

Velocidade - controla a velocidade de rotação da turbina, atuando sobre as válvulas e o distribuidor de modo a aumentar ou diminuir a quantidade de movimento transferido pela queda d'água

Reostatos de Ajuste de Referência - Servem para ajustar as diversas grandezas que são usadas pelos reguladores como referência

Tensão - determina em qual tensão o regulador de tensão deve manter a saída

Velocidade - idem para velocidade

Potência - idem para potência no regulador de velocidade

Limitador do Reg. de Velocidade - determina a abertura máxima e controla o fechamento do distribuidor em emergência

Sincronizador Automático - Sistema de malha fechada capaz de, atuando sobre os reguladores, levar a unidade ao sincronismo com o sistema elétrico. O sincronismo é obtido quando da igualdade dos valores de tensão e frequência entre a unidade e o sistema.

Freio - Sistema de sapatas com a finalidade de desacelerar a turbina até a parada total da mesma.

Comportas - Sistema de segurança para isolar o conduto forçado da tomada ou saída d'água.

Montante - isola o conduto forçado da tomada d'água

Jusante - isola o tubo de sucção da saída d'água

Síncrono-Gerador - Sistema de malha fechada capaz de levar a máquina do modo síncrono para o modo gerador e vice-versa. A máquina é dita no modo síncrono quando não há passagem de água no conduto forçado - válvula esférica fechada - e a mesma gira como um motor excitado pela rede elétrica.

3.3. REQUISITOS FUNCIONAIS DO SISTEMA DE CONTROLE

O sistema de controle deve ser capaz de atuar sobre os equipamentos acima descritos de forma a executar as sequências de comandos que permitirão a realização das seguintes manobras:

Partida Automática
Parada Automática
Parada parcial com rejeição de carga
Parada parcial sem rejeição de carga
Parada total com rejeição de carga
Parada total sem rejeição de carga
Transferência de modo (Gerador - Síncrono - Gerador)
Regulagem de Tensão
Regulagem de Velocidade e Potência
Comando sobre Disjuntores
Comando sobre Seccionadoras

O Sistema de Controle recebe as informações dos diversos sensores do processo. Eles indicam o estado das chaves e válvulas, níveis de pressão e de água, valores de tensão, corrente, ângulo de fase e etc.

A representação da Lógica do Sistema de Controle nada mais é do que uma descrição formal dos estados em que determinados sensores devem se encontrar para que um conjunto de comandos (saídas sobre o processo) deva ser executado.

Por exemplo, durante a sequência de Parada Automática, se o sensor de velocidade de rotação da turbina estiver ativo - indicando baixa rotação - e o sensor do distribuidor acusar seu fechamento, são executados os comandos para aplicação do freio e para acionamento da bomba de cunha de óleo.

Existem, atualmente, três formas de descrever a Lógica de Controle:

Diagramas Ladder

Diagramas Lógicos - Lógica Combinacional

Linguagem de Máquinas Sequenciais - MS

3.4. REPRESENTAÇÃO DA LÓGICA DE CONTROLE EM DIAGRAMAS LADDER

É a representação concreta da implementação da Lógica de Controle convencional, ou seja, apresenta, sem qualquer tipo de abstração, o Sistema de Controle da Unidade Geradora.

A representação em diagramas Ladder é a mais complexa de todas. Esta complexidade faz com que seja utilizada para documentação e manutenção do Sistema de Controle, preferindo-se a representação em Lógica Combinacional para a fase de projeto.

A dificuldade maior desta notação está no fato dos diagramas incluirem todos os detalhes de implementação da Lógica convencional, inclusive a numeração e posição física das conexões, relés, contatos e fios.

Outro agravante na complexidade é quanto à forma de realizar as funções de controle. Estas funções são implementadas com relés e seus contatos. Estes últimos

podem ser do tipo normalmente aberto ou normalmente fechado - 'NOT' lógico. Para se realizar, então, uma função lógica do tipo 'AND', conectam-se os contatos dos relés em série e para realizar um 'OR', conectam-se em paralelo.

As bobinas dos relés são excitadas pelos sensores do processo - p. ex. estado do disjuntor aberto, velocidade da máquina em 90%, etc - ou pelos próprios comandos que partem do Sistema de Controle.

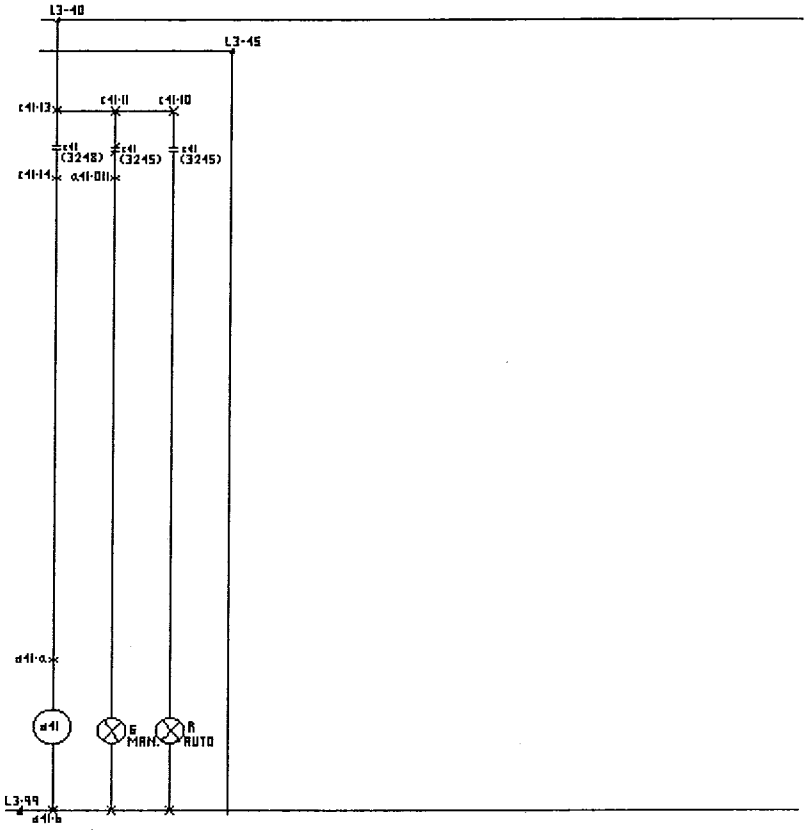


FIGURA 3.3 - DIAGRAMA LADDER

Este verdadeiro 'emaranhado' de simbolos de relés, contatos - um ou mais por relé - e fios é, para piorar, organizado em páginas, com referências cruzadas entre componentes de páginas diferentes. Tudo isso associado a uma notação que indica em que página de diagrama o componente aparece, o tipo de contato, em que posição física do painel ele se encontra, entre outras informações.

Se tomarmos como exemplo o trecho da sequência de Parada Automática que foi citado no ítem anterior, o diagrama Ladder deveria constar de dois contatos em série com duas bobinas de relés, também em série. Os contatos referentes à velocidade e ao fechamento do disjuntor deveriam indicar a que relés pertencem, relés estes que podem aparecer em outra página de diagrama. As bobinas ativam contatos que comandam o Freio e a Bomba de Cunha de Óleo e que também podem aparecer em uma ou várias páginas de diagrama.

A figura anterior apresenta um trecho de um diagrama Ladder de complexidade média onde podem-se visualizar todos os problemas mencionados.

3.5. REPRESENTAÇÃO DA LÓGICA DE CONTROLE EM LÓGICA COMBINACIONAL

A representação utilizando Diagramas Lógicos é uma forma abstrata de descrever o Sistema de Controle, que é, na realidade, implementado com lógica de relés (Diagrama Ladder). Por esta razão, esta representação só é utilizada durante o projeto da Lógica de Controle, sendo posteriormente implementada em Ladder. Este sim, é usado para manutenção, documentação e depuração da Lógica de Controle.

Deve-se destacar que a abstração alcançada com o modelo de representação Combinacional pouco ajuda o projetista de uma usina, de quem se exige um conhecimento intuitivo do controle, formalizado em Diagrama Lógico e depurado em Lógica de Relés. Além disto, este modelo não possui um formalismo puro. Elementos de temporização, que na prática são obtidos dos próprios relés (relés com contatos temporizados), são representados junto com elementos de memorização. São utilizados, ainda, simbolos de bobinas de relés cujos contatos, sem simbologia, servem como entradas para os operadores 'AND' e 'OR'.

Este Diagrama, assim como o anterior, guarda a notação para indicar a que relé um contato pertence, que página de diagrama ele aparece, entre outras coisas. Ele omite, apenas, informações quanto as disposições físicas dos componentes.

A figura 3.4 apresenta um trecho do Diagrama Combinacional para ilustração.

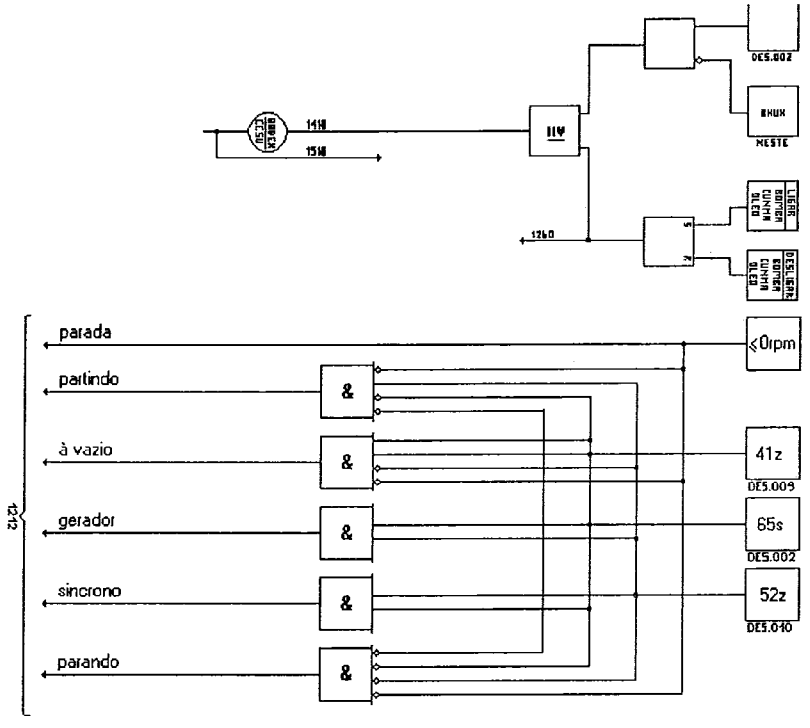


FIGURA 3.4 - DIAGRAMA LÓGICO

3.6. REPRESENTAÇÃO DA LÓGICA DE CONTROLE EM LINGUAGEM MS

Com o emprego de Sistemas Digitais em substituição aos Sistemas de Controle Convencionais, surgiram os CLP's - Controladores Lógico-Programáveis. Estes equipamentos permitem a sua programação de acordo com os requisitos do processo que irão controlar. A forma desta programação varia entre linguagens de

programação, linguagens orientadas a CLP's e, mais recentemente, sistemas especialistas para configuração de CLP's [S&WECo 89].

O Sequenciador de Partida, Parada e Operação de Unidades Geradoras - SEQ - é um exemplo, desenvolvido no Cepel, de um CLP com linguagem de programação própria [MIRAND 89]. Ele pode ser definido como uma máquina virtual para execução de autômatos finitos. Tais autômatos estão organizados numa estrutura hierarquizada, o que determina a ordem de execução. Na prática, estão definidos três níveis de hierarquia:

Nível Equipamento - neste nível estão todos os autômatos que controlam os equipamentos- p. ex. disjuntores, seccionadoras, bombas, etc.

Nível Sequência - neste estão os autômatos que realizam as sequências de controle dos equipamentos. Por exemplo, para realizar a parada da máquina deve-se emitir um controle sobre o Disjuntor de Campo, depois sobre a Bomba de Cunha de Óleo, depois sobre o Freio, etc...

Nível Comando - este é o nível mais alto da hierarquia. É sobre ele que o operador humano atua, disparando um solicitação para execução de uma manobra. Por exemplo, a Parada Automática é uma manobra que exige uma sequência de diminuição de carga, uma sequência de parada, etc.

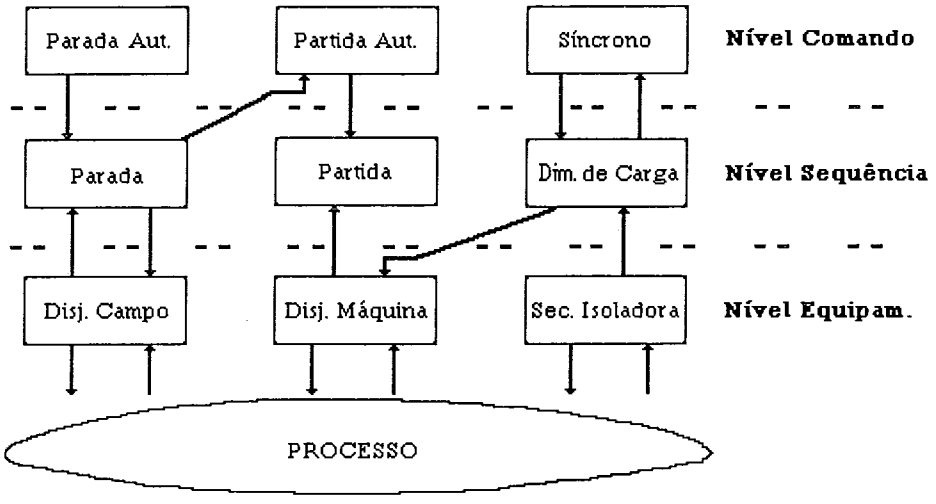
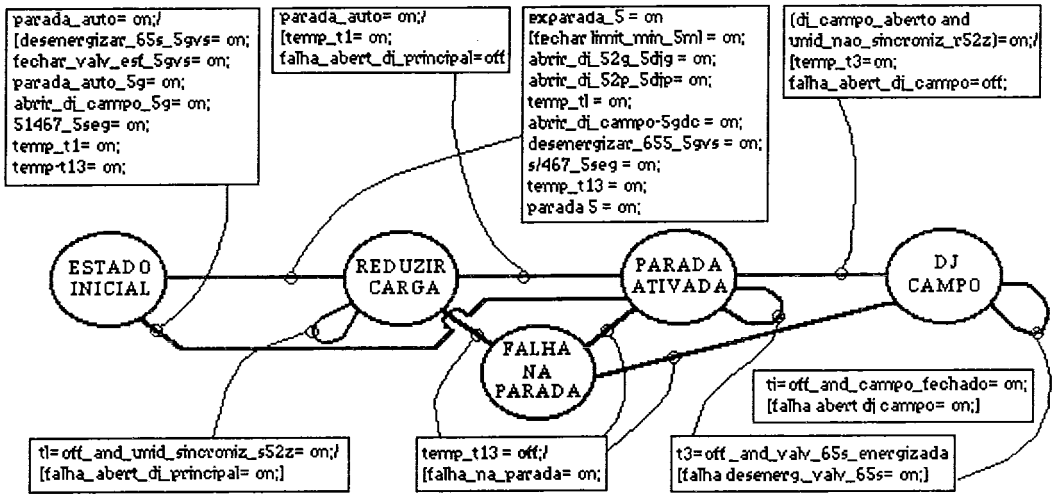


FIGURA 3.5 - ESTRUTURA DE COMUNICAÇÃO ENTRE AUTÔMATOS

Para se programar o SEQ foi desenvolvida uma linguagem de programação chamada MS - Máquinas Sequênciais. Esta linguagem permite que autômatos finitos sejam descritos sem nenhuma necessidade de adaptação do modelo para a implementação, evitando-se a introdução de erro nesta fase.

Em MS é permitida a comunicação entre autômatos de níveis diferentes. Esta comunicação é realizada através de variáveis booleanas chamadas 'Comandos' e 'Anunciadores'. Os Comandos pertencem a autômatos e só podem ser consultados por outros de nível inferior, enquanto que os Anunciadores são duais àqueles, ou seja, são ativados ou desativados por seus autômatos e só podem ser consultados por outros de nível superior. Existem, ainda, as entradas e saídas do processo, que funcionam como Anunciadores e Comandos, respectivamente, para os autômatos de nível mais baixo na hierarquia (Nível Equipamento).

A figura 3.6 apresenta o autômato de parada do Nível Sequência na sua representação gráfica e uma parte do programa MS correspontende.




```
COMPILADOR MS V1.0 - CEPEL/ELETROSUL      seq003.LMS
     PAGINA: 3

111
112  PARAUTO_EST_INIC : ESTADO;
113
114            PARA PARAUTO_ RED_CARG
115                  SE EX_PARADA_5 = ON
116
117                     [ FECHAR_LIMIT_MIN_EML = ON ;
118                       ABRIR_DJ52G_5DJG = ON;
119                       ABRIR_DJ52P_5DJP = ON;
120                       ABRIR_DJCAMPO_5GDC = ON;
121                       DESENERGIZAR_65S_5GVS = ON ;
122                       FECHAR_VALV_ESF_5GVE = ON ;
123                       S1467_5SEQ = ON ;
124                       PARADA_5 = ON ;
125                       TCL_PARADA_AUTO = OFF ;
126                       TSABRJU = ON ;
127                       TSPARTOT = ON ; ]
128
129            PARA PARAUTO_PRD_ATIV
130                  SE PARADA_AUTO = ON
131
132                     [ ABRIR_DJCAMPO_5GDC = ON ;
133                       DESENERGIZAR_65S_5GVS = ON ;
134                       FECHAR_VALV_ESF_5GVE = ON ;
135                       S1467_5SEQ = ON ;
136                       PARADA_AUTO_5G = ON ;
137                       TCL_PARADA_AUTO = OFF ;
138                       TSABRDJC = ON ;
139                       TSPARTOT = ON ;
140
141  FIM PARAUTO_EST_INIC ;
```

FIGURA 3.6 - AUTÔMATO DE SEQUÊNCIA DE PARADA

CAPÍTULO IV

SOLUÇÃO PROPOSTA

Todas as representações da Lógica de Controle que foram apresentadas exigem do projetista uma tradução do modelo intuitivo, que ele possui do processo, para o modelo formal em que se baseia a representação.

Para realizar esta tradução o especialista no processo precisa conhecer bem este modelo formal e se ater aos detalhes da implementação. Esta é, sem dúvida, uma tarefa complexa e sujeita a erro, além de consumir muito tempo do projeto e desenvolvimento da Lógica de Controle.

A proposta deste trabalho é eliminar esta tradução. Fornecer um sistema inteligente, capaz de, a partir de uma interface em Linguagem Natural, interagir com o projetista no processo de descrever a Lógica de Controle bem como de depurá-la.

A proposta apresentada é criar um ambiente de desenvolvimento para Sequenciadores. Este ambiente está dividido em dois sistemas: o Sistema de Desenvolvimento propriamente dito e o Sistema de Depuração e Testes.

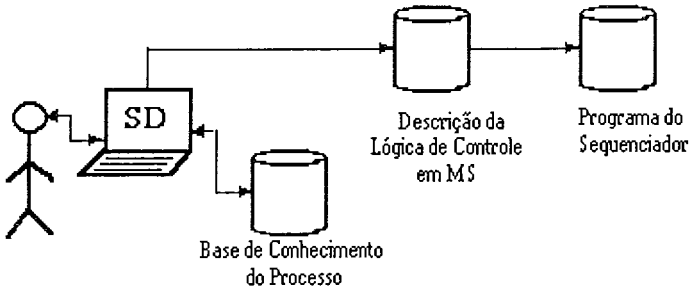


FIGURA 4.1 - SISTEMA DE DESENVOLVIMENTO

4.1. SISTEMA DE DESENVOLVIMENTO

O Sistema de Desenvolvimento é um sistema interativo para aquisição de conhecimento do especialista no processo de uma usina hidrelétrica.

A partir de uma base de conhecimento genérica sobre usinas e os elementos que as compõem, o especialista descreve, em Linguagem Natural, a lógica de controle da usina. Esta descrição é feita através de sentenças em português que podem ser questionadas ou rejeitadas pelo sistema com o intuito de se extrair, com segurança, as afirmativas nelas contidas.

ESP >	QUANDO O FREIO É APLICADO, A VELOCIDADE DA MÁQUINA DEVE SER MENOR QUE X rpm.
SD >	A APLICAÇÃO DO FREIO É DO TIPO CONTÍNUA OU TEMPORIZADA?
ESP >	CONTÍNUA.

FIGURA 4.2 - DIÁLOGO ESPECIALISTA - SISTEMA DE DESENVOLVIMENTO

Uma vez detalhada toda a lógica de controle, o sistema gera uma descrição desta em uma Linguagem Artificial qualquer. Esta descrição é submetida a um compilador gerando assim o programa executável do Sequenciador.

Por uma questão de disponibilidade de ferramentas para o desenvolvimento deste programa executável e do poder de representação da lógica num nível de compreensão razoável, foi escolhido, como Linguagem Artificial, o MS.

4.2. SISTEMA DE DEPURAÇÃO E TESTES

É através do Sistema de Depuração e Testes que o especialista valida o programa executável que foi criado com o SD.

O programa do Sequenciador é carregado no hardware específico e é posto para rodar. O Sistema de Depuração e Testes se liga à rede do processo para ter acesso a todas as informações que as réplicas do Sequenciador possuem. Com os dados do processo e com a base de conhecimento que foi criada no Sistema de Desenvolvimento, o SDT pode acompanhar cada passo de decisão que está sendo tomada no Sequenciador, mostrando para o especialista como, porque e o que se passou na lógica de controle da unidade geradora.

O SDT pode também simular comandos que partiriam normalmente da Sala de Controle do Sistema Distribuído de Controle Digital (SDCD) da usina, gerando assim todos os estados possíveis da lógica para teste.

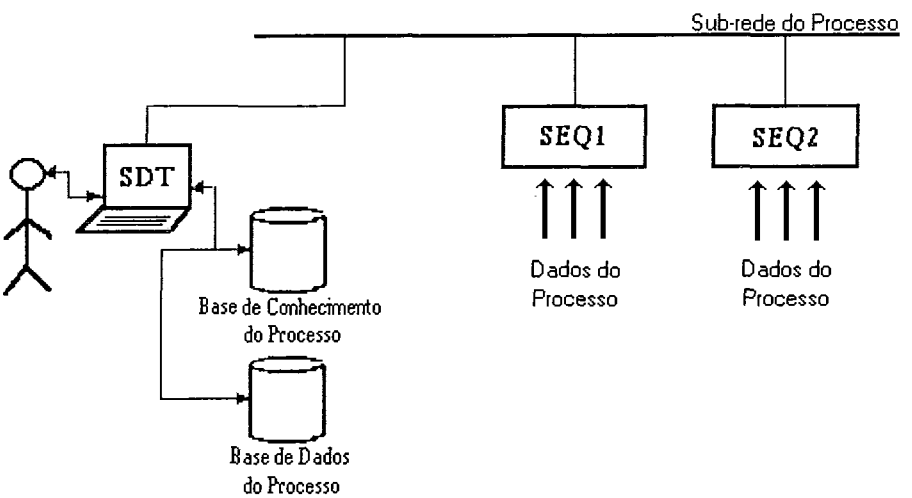


FIGURA 4.3 - SISTEMA DE DEPURAÇÃO E TESTES

CAPÍTULO V

PROCESSAMENTO DE LINGUAGEM NATURAL

Segundo [HAYES 83], Processamento de Linguagem Natural - PLN - pode ser definido como: "A formulação e investigação dos mecanismos computacionais efetivos para comunicação via Linguagem Natural".

Em outras palavras, é uma área que trata do uso da Linguagem do ser humano, como Português, Inglês, Espanhol, etc, como meio de comunicação entre uma pessoa e o computador. Por isso, PLN não estuda a LN num nível abstrato, mas enfocando a obtenção de mecanismos que tornem viável a construção de programas que realizem ou simulem uma comunicação em LN. Esta caracteristica a diferencia da Linguistica, assim como de outras disciplinas que estudam LN.

A Linguistica por exemplo, está concentrada em encontrar modelos formais que generalizem as LN's. O objetivo é conseguir modelos que capturem, o máximo possível, as regularidades da Linguagem, fazendo as generalizações linguisticas apropriadas.

A Psicologia, na área da Ciência da Cognição, é outra disciplina que estuda LN. Esta, assim como a Linguistica, não busca aplicação computacional, mas apenas modelos que determinem como usamos a LN e que tornem possível compreender o raciocínio humano e sua organização mental.

A própria área de PLN possui dois enfoques diferentes e que podem ser classificados como: Processamento de Linguagem Natural Aplicada e Processamento de Linguagem Natural em Geral. Este último tem por objetivo tomar os estudos da Psicologia do ponto de vista da Ciência da Computação. Os trabalhos nesta área, como [SCHANK 75], [CULLIN 78] entre outros, tratam de sistemas para compreenção de textos ou estórias.

Uma das principais lições que foram tiradas é que PLN em geral requer uma grande quantidade de conhecimento do mundo real. E tais trabalhos dedicaram bastante esforço na tentativa de representação deste mundo. Por ainda não serem tratáveis computacionalmente, os sistemas que surgiram destes trabalhos servem apenas como demostrações da factibilidade dos conceitos ou abordagens propostos.

O PLN aplicada é uma área preocupada em apenas permitir que os computadores se comuniquem em LN. A ênfase e totalmente pragmática, sendo menos importante se um computador 'entende' a entrada em LN, mas indispensável que ele responda a ela de forma cooperativa e de acordo com os objetivos expressos na mesma.

5.1. PROBLEMAS BÁSICOS NO PROCESSAMENTO DE LN

Existe uma palavra que praticamente traduz todos os problemas no PLN: ambiguidade. Ela pode ocorrer de várias formas.

- Sintática ou Estrutural

  João viu a mulher de binóculos(quem usava binóculos?)

- Significado da Palavra

  A minha manga rasgou/apodreceu

- Caso

  Eu corri 10Km em uma hora/Brasília (adjunto adverbial de tempo/lugar)

- Referencial

  Eu tirei a maça da torta e a comi (o que foi comido, a torta ou a maça?)

Um sistema de PLN precisa resolver tais ambiguidades antes de traduzir a sentença para uma representação interna única. O sistema precisa, então, inferir sobre as possibilidades de forma a escolher a correta. Isto é feito baseado no conhecimento que ele possui da aplicação. Se o domínio da aplicação engloba o

mundo real, torna-se muito dificil uma inferência. Quando o domínio diz respeito a um campo específico do conhecimento a solução é bem mais simples de ser alcançada. No contexto de Usinas e seu Sistema de Controle, as ambiguidades da sentença abaixo, seriam facilmente retiradas.

Se a turbina estiver parando e a chave seccionadora estiver fechada, abra-a.

Ambiguidade -	Inferência -
Significado da palavra chave	Só existe um tipo de chave na Usina
Referencial - pronome 'a'	Só é possível abrir uma chave, nunca uma turbina

5.2. TÉCNICAS PARA ANÁLISE DE LINGUAGEM NATURAL

Por Análise de Linguagem Natural entende-se como sendo o mecanismo de tradução das sentenças de entrada para sua representação interna no sistema de PLN.

Existem algumas abordagens ou classificações quanto a forma de se realizar a Análise de Linguagem Natural. Nesta sessão são mencionadas algumas das mais comuns.

5.2.1. Casamento de Padrões - (Pattern Matching)

A essência nesta abordagem é interpretar a sentença como um todo, sem se preocupar com os seus constituintes. Para isso o sistema procura associar a entrada com um de seus padrões pre-definidos de sentenças. Para cada padrão existe uma interpretação correspondente.

Em [WEIZEN 66] é utilizada esta abordagem na construção do sistema ELIZA. A função deste sistema era simular um psicanalista da escola Rogeriana durante a entrevista com um paciente. Este estilo de psicanálise permite ao sistema responder convenientemente, apesar de não haver um 'entendimente' real do dialogo. Este entendimento não era alcançado pois as sentenças não eram convertidas para nenhuma representação interna, mas, simplesmente, disparavam respostas selecionadas a partir de um conjunto de padrões predefinidos de entradas. Um exemplo de padrões e respostas de ELIZA seria:

PADRÕES	RESPOSTAS
X sempre *Y*	-Você poderia ser mais específico? -É mesmo? -Realmente, sempre?
X você *Y* mim	-Por que você acha que alguém *Y* você?
X W me *Y*	-Suponha que *W* realmente te*Y*, o que isso significaria para você? -Você gosta de pensar que todo mundo te*Y*, não é?

DIÁLOGO

P: As pessoas sempre riem de mim.	- Primeiro padrão
E: Você pode ser mais específico ?	
P: Por exemplo você sempre ri de mim.	- Segundo padrão *Y* = 'sempre ri de'
E: Por que você acha que alguém sempre ri de você?	
P: Porque esse alguém me odeia	- Terceiro padrão *Y* = 'odeia'
E: Você gosta de pensar que todo mundo te odeia, não é?	

ELIZA demonstrou como um mecanismo simples era capaz de imitar um comportamento aparentemente inteligente.

5.2.2. Analise Orientada à Sintaxe - (Phrase Structured)

Nesta abordagem utiliza-se a analise sintática como forma de interpretar a sentença. Esta é desmembrada nos seus constituintes imediatos - Sintagmas Nominais, Verbais, etc - que por sua vez também são desmembrados, seguindo iterativamente até se obter as palavras isoladas. Somente após se construir toda a estrutura da sentença, o sistema a traduz para a sua representação interna.[6]

As técnicas mais conhecidas que seguem esta abordagem são: DCG - Definite Clause Grammar [PEREIR 80] e ATN - Augmented Transition Networks [WOODS 70]. Estas técnicas são comparativamente analisadas com RVG - Register Vector Grammar [BLANK 85] [BLANK 89] no final deste capítulo.

5.2.3. Gramáticas Semânticas - (Semantic Grammars)

Esta abordagem é essencialmente igual a anterior, diferindo apenas, no fato das categorias utilizadas serem definidas semanticamente, ao invés, ou além, de sintaticamente.

Logo, no lugar da categoria 'Sintagma Nominal' uma gramática semântica teria, p. ex., a categoria 'Descrição da Partida' que seria sempre preenchida pela categoria sintática 'Sintagma Nominal'.

Esta abordagem foi primeiro proposta por Burton para aplicação no SOPHIE [BROWN 75].

5.2.4. Gramática de Casos

Foi proposta por Fillmore [FILLMO 68] e logo adotada por vários pesquisadores, entre eles Schank dentro do modelo de Dependência Conceitual [SCHANK 75].

---

[6] existem casos onde a interpretação é feita simultaneamente com a análise, o objetivo é resolver alguns tipos de ambiguidades a partir de inferências sobre a interpretação

A ideia básica é a criação de conceitos ou papeis principais associados com um conjunto de conceitos ou papeis secundários. Por exemplo:

João quebrou a vidraça com um martelo na rua ao lado.

[Quebrar
  [Casos
    Agente - *João*
    Objeto - *vidraça*
    Instrumento - *martelo*
    Recipiente -
    Diretivo -
    Locativo - *na rua ao lado*
    Co-agente -]
  [Modos
    Tempo - *Passado*
    Voz - *Ativa*]]

Na Gramática de Casos a relação entre a cabeça (papel principal) e os casos (agente, objeto, etc) é estabelecida semanticamente. Caso a frase fosse construida com *martelo* realizando a função sintática de sujeito (*O martelo quebrou a vidraça ...*) ou com *vidraça* como sujeito (*A vidraça foi quebrada ...*), a atribuição dos papeis, ou casos, continuaria a mesma, independente das funções sintáticas das palavras terem mudado.

Por tanto, para se analisar LN utilizando Gramática de Casos é preciso um conhecimento semântico da aplicação, além de informações sobre a sintaxe.

O modelo de Dependência Conceitual do Schank é uma forma extremamente canônica de se tratar a Gramática de Casos. Entendendo-se por canônica como uma forma de representação onde maneiras diferentes de se expressar a mesma informação tem representações idênticas. Proposições que

enfocam informações similares são representadas de forma intencionalmente parecidas mas que explicitem as diferenças.

Na Dependência Conceitual as ações são agrupadas por tipos bastante genéricos como por exemplo:

ATRANS (Abstract TRANSfer) - dar, tomar, tirar, etc,

PTRANS (Physical TRANSfer) - correr, jogar, etc,

MTRANS (Mental TRANSfer) - dizer, contar, relatar, etc,

MBUILD (Mental BUILD) - realizar, concluir, imaginar, etc,

ATTEND (ATTEND) - olhar, observar, ouvir, assistir, etc.

5.3. TÉCNICAS DE ANÁLISE SINTÁTICA

O dilema traçado no início do capítulo entre PLN, com abordagem preocupada com a complexidade computacional, e a generalidade buscada pela Linguística, obteve uma solução quando Chomsky afirmou que para se alcançar a competência para se processar LN, um analisador sintático requer, pelo menos, o poder de uma gramática livre de contexto. Só assim, ele será capaz de tratar as estruturas mais comuns da linguagem. Ele propôs, ainda, o uso de transformações na estrutura das frases para se obter o mesmo resultado da análise independentemente do posicionamento das palavras - Gramática Transformacional.

Duas técnicas vem comumente sendo usadas em PLN: DCG e ATN. Ambas satisfazem o requisito proposto por Chomsky, sendo ambas equivalentes a uma Máquina de Turing em poder de representação.

No entanto, surgiu recentemente uma técnica chamada RVG - Register Vector Grammar - que apesar de equivaler a uma gramática regular, se mostra muito eficiente para PLN. A seguir, são apresentadas estas técnicas e, no próximo capítulo, algumas sujestões de modificação no RVG que o torna tão completo quanto ATN e bem mais compacto.

5.3.1. Definite Clause Grammar - DCG

A forma correta de se tornar precisa a definição de uma linguagem, natural ou artificial, é através da utilização de um conjunto de regras chamado gramática. Estas regras de gramática definem quais sequências de palavras são sentenças válidas da linguagem e quais não são.

As diferentes formas ou complexidade das regras definem classes diferentes de gramática [CHOMSK 57] e uma classe fundamental e bastante familiar para comunidade informática na forma BNF (Bacus-Naur Form) é a da Gramática Livre de Contexto (GLC) (Apêndice A). Em GLCs, as palavras ou símbolos básicos da linguagem são identificados por *símbolos terminais* enquanto categorias de frases ou de palavras, por *símbolos não-terminais.* Cada regra da GLC expressa a possibilidade de se formar um único símbolo terminal pela ocorrência de um ou mais símbolos terminais ou não-terminais.

Em 1974, Colmerauer e Kowalsky tiveram a idéia de traduzir o formalismo de GLC para um formalismo bem mais genérico; Cálculo de Predicados de Primeira Ordem. Eles desenvolveram um método particular de expressar regras Livre de Contexto como sentenças Lógicas de um tipo restrito, conhecido como *Cláusulas Definidas* (CD) ou *Cláusulas de Horn*[7] (CH). O problema de analisar uma sentença é, assim, transformado no problema de provar que um certo teorema é derivado de um conjunto axiomático de Cláusulas Definidas que descrevem a Linguagem.

Esta idéia não despertaria mais do que interesse teórico, não fosse o fato de Colmerauer e Kowalsky, na mesma época, terem sugerido que uma coleção de regras em CD pode ser considerada um programa. Isto significa que a dedução

[7] Cláusulas Definidas são um tipo de Cláusulas de Horn onde se tem uma e sempre uma cabeça(lado esquerdo) na cláusula

automática possui todas as características que se associa à computação e aos seus métodos de execução.

Um resultado prático destes conceitos de 'Programação em Lógica' foi obtido por Colmerauer et alii na forma da Linguagem de programação Prolog [PEREIR 80].

A GLC expressa em CD, de acordo com o método de COlmerauer-Kowalsky, e executada como um programa Prolog se comporta como um analisador sintático para a linguagem que a GLC descreve. Este fato é ainda mais importante se observado que a técnica de traduzir GLC em CD resulta num formalismo muito mais poderoso que as GLC, e ainda executável em Prolog. Este formalismo é DCG (Definite Clause Grammar).

Uma vez que GLC não são totalmente adequadas para PLN, DCG supera as limitações aumentando a GLC em três sentidos importantes. Primeiro, DCG provê a gramática com dependências de contexto, permitindo que a forma dos constituintes da sentença dependa do contexto no qual este constituinte ocorre dentro da sentença.

Segundo, permitindo que estruturas em árvore possam ser construídas durante a análise de uma maneira não limitada pela estrutura recursiva da gramática.

E por último, DCG permite que condições extras sejam incluídas nas regras da gramática, fazendo com que o rumo da análise dependa de uma computação auxiliar com poder ilimitado.

Para descrever como a gramática pode ser expressa em Lógica, considere a notação para GLC abaixo:

nt $-\triangleright$ *corpo.*

onde *nt* é um símbolo não-terminal e corpo é uma sequência de um ou mais ítens separados por vírgulas. Cada ítem pode ser um símbolo terminal ou não-

terminal. O significado para esta regra é: *'corpo* é uma forma possível do constituinte de tipo *nt'*. Outras formas possíveis de *nt* devem ser declaradas e todos os demais símbolos não-terminais devem ter, pelo menos, uma cláusula de definição.

GRAMÁTICA LIVRE DE CONTEXTO		
Sentença	—▷	SN, SV
SN	—▷	artigo, substantivo, Cls_Relativ
SN	—▷	nome_próprio
SV	—▷	verbo_trans, SN
SV	—▷	verbo_intrans
Cls_Relativa	—▷	[que], SV
Cls_Relativa	—▷	[]
artigo	—▷	[todo]
artigo	—▷	[uma]
substantivo	—▷	[homem]
substantivo	—▷	[mulher]
nome_próprio	—▷	[João]
nome_próprio	—▷	[Maria]
verbo_trans	—▷	[ama]
verbo_intrans	—▷	[vive]

TABELA 5.1 - GLC PARA SENTENÇAS SIMPLES

A tabela acima apresenta uma GLC para ilustração. A linguagem definida nesta figura cobre sentenças do tipo: *João ama Maria* ou *Todo homem que vive ama um mulher.*

Para traduzir esta gramática deve-se associar cada símbolo não-terminal a um predicado de dois termos com o mesmo nome do símbolo. Cada termo deste predicado guarda a posição de início e fim, na sentença, do constituinte que satisfaz o predicado.

A tabela 5.2 apresenta a mesma GLC em DCG. Nesta forma, a primeira cláusula pode ser lida como: 'a sentença estende-se de S0 a S se existe um Sintagma Nominal entre S0 e S1 e um Sintagma Verbal entre S1 e S'. Ainda, a sétima cláusula seria lida como: 'uma cláusula relativa se estende de S a S', ou seja, vazia.

Para se representar os símbolos terminais, usa-se o predicado *conecta(S1, T, S2),* onde se entende: 'o símbolo T estende-se de S1 a S2 na sentença'.

DEFINITE CLAUSE GRAMMAR	
Sentença(S0, S)	:- SN(S0, S1), SV(S1, S).
SN(S0,S)	:- artigo(S0,S1), subst(S1,S2), c_rel(S2,S)
SN(S0,S)	:- subst(S0,S).
SV(S0,S)	:- verbo_trans(S0,S1), SN(S1,S).
SV(S0,S)	:- verbo_intr(S0,S).
c_rel(S0,S)	:- conecta(S0, "que",S1), SV(S1,S).
c_rel(S,S)	:- .
artigo(S0,S)	:- conecta(S0, todo, S)
artigo(S0,S)	:- conecta(S0, uma, S)
subst(S0,S)	:- conecta(S0, homem, S)
subst(S0,S)	:- conecta(S0, mulher, S)
nome_pró(S0,S)	:- conecta(S0, João, S)
nome_pró(S0,S)	:- conecta(S0, Maria, S)
verbo_trans(S0,S)	:- conecta(S0, ama, S)
verbo_intr(S0,S)	:- conecta(S0, vive, S)

TABELA 5.2 - DCG PARA SENTENÇAS SIMPLES

Sensibilidade ao Contexto

Para representar gramáticas sensíveis ao contexto em DCG, basta aumentar o número de termos nos predicados, de forma a armazenar e transmitir este contexto na forma de variáveis. Por exemplo, a concordância de número entre verbo

e sujeito num sentença é obtida pela colocação de um termo N, que assume os valores *Singular e Plural*, em todas as cláusulas da tabela anterior.

A associação de um termo N num predicado com outro termo N em outro predicado é automática em Prolog e faz parte do processo de Unificação da linguagem [CASANO 86].

5.3.2. Augmented Transition Networks - ATN

ATN é um formalismo baseado em RTN que por sua vez possui o poder de uma gramática livre de contexto. As mudanças introduzidas por ATN aumentam este poder ilimitadamente (Gramática Irrestrita), dando a este a possibilidade de tratar todas as formas da Linguagem Natural.

As mudanças se resumem ao uso de registros e de operações, de teste e atribuição destes registros, que estão associadas a cada transição da rede. Os registros são como variáveis das linguagens de programação, cada uma com um nome e armazenando uma certa informação. As operações de teste ou Condição restringem as circunstâncias sob as quais uma transição pode ocorrer, enquanto as atribuições ou Ações montam estruturas com os registros ou associam valores aos seus campos de atributo. As Condições podem depender de constituintes que já ocorreram, de propriedades especiais da palavra/constituinte que irá satisfazer o arco ou, ainda, de propriedades da estrutura que foi construída até o momento.

Os registros são compostos por campos de *atributos* e de *papéis*. Um campo de atributo armazena um determinada característica da palavra ou constituinte, enquanto o campo de papel aponta para outro registro que desempenha o "papel" associado ao campo. Por exemplo (figura 5.1), uma sentença composta de Sintagma Nominal e Verbal tem seu registro no campo de *papel de Sujeito* apontando para o registro de Sintagma Nominal, enquanto o campo de atributo de *Voz* é preenchido

com *Passiva*. Na representação gráfica da figura uma linha horizontal divide os campos de atributo dos campos de papel.

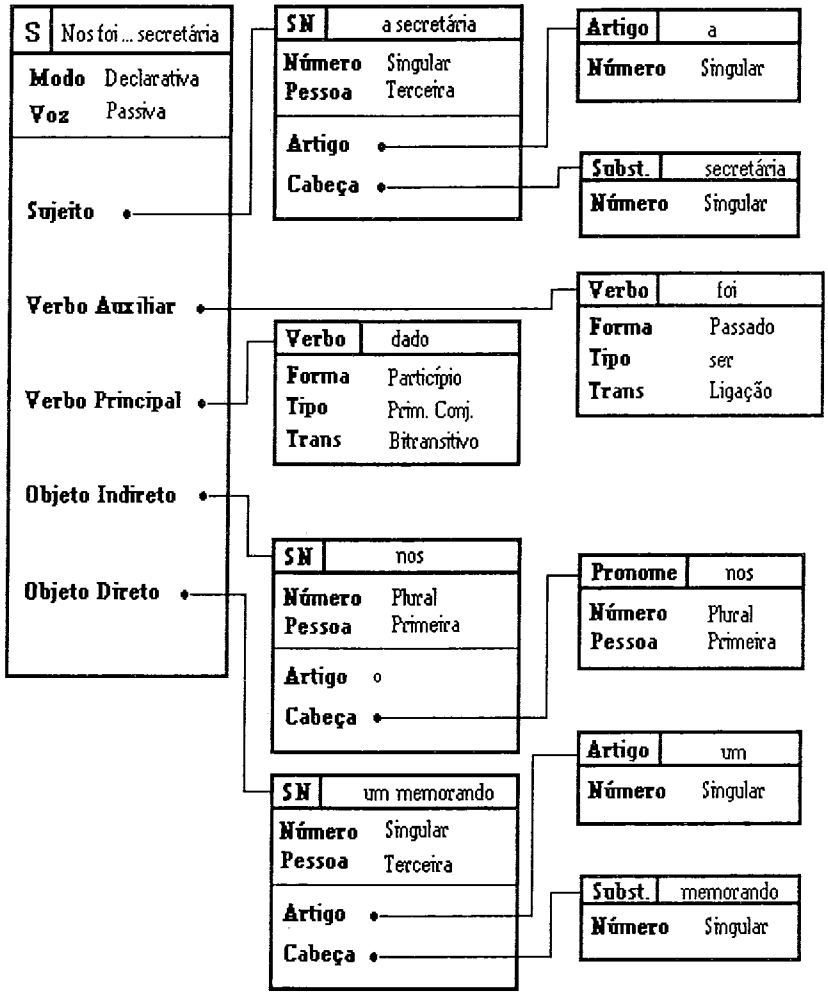


FIGURA 5.1 - ESTRUTURA DE REGISTROS NO ATN

Em resumo, as modificações introduzidas por ATN são:

Registro - são variáveis que armazenam valores atribuídos durante a análise. Possui dois campos distintos: papel e atributo. O campo de papel identifica papéis desempenhados por determindados constituintes da frase. O campo de atributo guarda determinadas propriedades da palavra ou constituinte.

Condições/Ações- estão associadas a cada arco (transição) da RT. Para que determinado arco seja percorrido, as Condições associadas devem ser satisfeitas. Quando um arco é percorrido, as Ações são executadas, fazendo com que campos dos registros sejam atualizados.

As Ações possíveis são:

a) Um campo de atributo é preenchido com uma das possibilidades .
b) Um campo de papel é apontado para outro registro.
c) Um registro é adcionado ao fim de uma sequência de registros.

As Condições testáveis são:

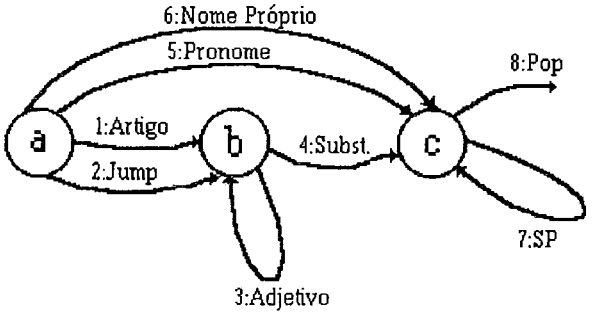
a) Dois campos de atributos são identicos
b) Um campo de papel está vazio
c) Um registro é de determinada categoria
d) Uma palavra associada ao registro é uma palavra específica
e) Qualquer combinação Lógica das anteriores

Iniciações - são Ações executadas no início da análise de uma sub-rede de transição e se pretam a preencher os campos dos registros com valores iniciais em função da análise até o momento.

Classe de Arcos- existem quatro Classes de Arcos diferentes. Um Arco de Categoria é aquele que precisa concordar a categoria da palavra ou constituinte que está sendo analisado com a sua própria categoria. Um Arco 'Push' é aquele que dispara a análise dee uma sub-rede de transição (recursão). Um Arco 'Send' ou 'Pop' é o arco de retorno à rede de transição que percorreu o 'Push'. O último tipo de arco, 'Jump', é percorrido sem o consumo de palavras ou constituintes da sentença de entrada. Todos os Arcos , conforme já foi dito, podem ter Ações/Condições associadas a si.

O Arco Jump atribui ao ATN um comportamento não deterministico. Este fato associado com a própria caracteristica das Ações/Condições é que dão a este formalismo a complexidade elevada que ele apresenta.

A figura 5.2 apresenta uma rede ATN completa para análise de um Sintagma Nominal. Nesta rede é feita a concordância de número entre Adjetivo, Substantivo e Artigo.



$_{a}$Artigo$_{b}$
A:Preencher Número com Número de *
$_{b}$Substantivo$_{c}$
C:Número está vazio ou é igual à Número de *
A:Preencher Número com Número de *
$_{b}$Adjetivo$_{b}$
C:Número está vazio ou é igual à Número de *
A:Preencher Número com Número de *
$_{a}$Pronome$_{c}$
A:Preencher Número com Número de *
$_{a}$NomPróprio$_{c}$
A:Preencher Número com Número de *

FIGURA 5.2 -REDE ATN PARA SINTAGMAS NOMINAIS

5.3.3. Register Vector Grammar - RVG

RVG ou Gramática de Vetores de Registro foi proposta em 1985 por Glenn D. Blank, [BLANK 85], e era, na sua definição inicial, equivalente a um autômato de estados finito - AEF - não determinístico. Blank afirma em seu artigo, que RVG é um "um autômato sensível ao contexto, sem ter, no entanto, o poder nem a complexidade de uma gramática sensível ao contexto, segundo a hierarquia de Chomsky". Esta característica da Gramática permite o tratamento de dependências e permutações da sintaxe (concordâncias, frases embutidas, etc) sem prejuízo na complexidade algoritmica de um analisador baseado em RVG. É garantida, então,

uma complexidade de tempo linear, ou seja, O(n). Uma melhoria significativa comparada com os tempos $O(n^3)$ obtidos em gramáticas livre de contexto [EARLY 70], [TOMITA 85] e [REED 87].

A posição defendida por Blank é de que um bom desempenho pode ser alcançado por um modelo com espaço finito de memória. E se tal modelo não é capaz de representar uma determinada construção gramatical, certamente um orador ou ouvinte desta linguagem não a utiliza, pois também não é capaz de compreendê-la. Conforme será abordado, sentenças embutidas como : *O rato que o gato que o cão mordeu perseguiu fugiu;* só são permitidas em RVG até um certo nível de recorrência.

5.3.3.1. A Gramática

Conforme já foi mencionado, RVG é equivalente a um autômato de estados finito - AEF - com duas inovações que trazem uma eficiência e compactação bem maiores que em AEF simples.

A primeira inovação está no uso de vetores para representar os estados do autômato. Em vez de sofisticar os tipos de arcos (categorias) - como ATN e DCG, com elementos não terminais, registros, etc - RVG abstrai nos símbolos de estado que passam a ser vetores. Estes vetores de estado compõem, em cada uma das suas dimensões, uma categoria da Gramática, eliminando uma série de redundâncias da gramática e tornando-a muito compacta. Os tipos de categoria são: léxica (subst, verbo), sintática (sujeito, objeto) ou qualquer (sintagma nominal, grupo adverbial, flag de ocorrência de outra categoria etc).

A segunda está nos Registros de Estado Corrente que permitem a escolha de estados alternativos, gerados por ambiguidades estruturais, e a retomada desta escolha por outra. Pode-se assim, dispensar o uso de "backtracking", bastando para

isso, reutilizar um registro já ocupado. Um número finito e pequeno destes registros garante complexidade de tempo linear com o tamanho da gramática.

Existe também um vetor de Condição e outro de Resultado associados com cada produção da gramática. Em AEF o "match" de uma determinada categoria de arco é obtido pela simples identidade desta com a categoria da palavra analisada. O resultado é a atualização do Estado Sintático Corrente com o novo estado do autômato. Já em RVG, o "match" é alcançado se, além da identidade das categorias, o Registro de Estado Sintático Corrente - RESC, que passa a ser um conjunto de vetores ternários, satisfizer o vetor Condição associado à produção da categoria. Um novo RESC é retirado da operação do antigo com o vetor Resultado, que também está associado à produção da categoria.

Formalmente, RVG é um autômato com uma quintupla (E,C,I,F,T) onde, E é o conjunto de estados, C o conjunto de categorias ou simbolos de entrada, I o estado inicial, F o conjunto de estados finais e T a função de transição que mapeia E x C em E.

Os estados e transições em RVG são representados por vetores de atributos com valores ternários. Os valores possíveis dos atributos são: + ou 1, que significa ligado ou habilitado; - ou 0, desligado ou desabilitado; ? ou 2, 'não importa'. Cada dimensão do vetor é chamada de atributo ou característica e está associada a uma categoria ou conjunto de categorias da Gramática.

A função de transição está baseada em duas operações sobre os vetores de atributos ternários: "match" e "change".

Dados f e g, dois vetores de atributos, a operação match produz um resultado Booleano do tipo:

$$\text{match } (f_i, g_i) \begin{cases} \text{verdade se } f_i = g_i \text{ ou } f_i = ? \text{ ou } g_i = ? \\ \text{falso caso contrario} \end{cases}$$

$$\text{match}(f, g) \begin{cases} \text{verdade se match}(f_i, g_i) \text{ para todos i's} \\ \text{falso caso contrario} \end{cases}$$

Ex: match ((+,-,-,?), (+,?,-,+)) é verdadeiro

match ((+,-,-,-), (+,-,-,+)) é falso

O operador change toma dois vetores de atributos f e g e os transforma em um terceiro da seguinte maneira:

$$\text{change}(f_i, g_i) = \begin{cases} g_i \text{ se } g_i = + \text{ ou } g_i = - \\ f_i \text{ se } g_i = ? \end{cases}$$

change (f, g) = change (f_i, g_i) para todos i's

Ex: change ((+,-,-,?,?), (?,+,-,?,-)) = (+,+,-,?,-)

A função de transição é implementada com três estruturas de dados:

Tabelas de Produções - quintuplas de (cat, cond, result, tipo_lex, ação) onde cat é um simbolo representando uma das categorias possíveis, cond e result são vetores de atributos, tipo_lex é um caractere para diferenciar os tipos de produção (inicial, final, léxica e não-lexica) e ação é a lista de funções executáveis sobre RESC.

Dicionário - duplas de (palavra, lista_cat) onde palavra é a descrição morfológica da palavra e lista_cat é a lista de categorias às quais a palavra pode pertencer.

Vetor de Estado - é um dos registros do RESC, ou seja, o vetor de atributos que a função de transição satisfaz e atualiza.

A função de transição do RVG é - dados o conjunto de vetores de estado E, o conjunto de simbolos de categorias C (contidos no dicionário) e o conjunto de produções - uma função T de E x C em E tal que o resultado de T seja obtido por uma produção (cat, cond, result, tipo_lex, ação) da seguinte maneira:

i) match(cond, RESC) é verdadeiro

ii) cat = uma das categorias em lista_cat

iii) RESC := change (RESC, result)

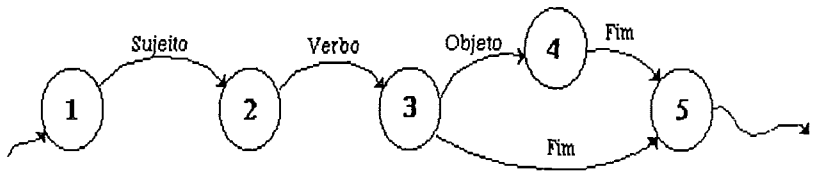
5.3.3.2. Algumas Propriedades

Uma das maiores vantagens de RVG está na sua compacidade diante de autômatos de estados finito ou mesmo recursivos aumentados como ATN.

Esta forma compacta pode ser melhor compreendida no exemplo da linguagem Sujeito, Verbo e Objeto a seguir.

Um autômato para esta linguagem SVO pode ser visto na figura 5.3a, enquanto a figura 5.3b apresenta a gramática correspondente em RVG. Pode-se observar que a ocorrência ou não do objeto (verbo transitivo ou intransitivo) precisa ser representada por duas transições com a categoria *Fim* no autômato finito. Em RVG, esta propriedade da linguagem é obtida pelo uso do valor "?" no atributo do

vetor que corresponde a categoria *Objeto*. Em outras palavras, o vetor *cond* da produção *Fim* é (-,-,?) satisfazendo tanto (-,-,+) como (-,-,-), ou seja, a ocorrência ou não do objeto.



a) REPRESENTAÇÃO EM AEF

Produções

Cat	Cond	Result.
Sujeito	+??	-??
Verbo	-+?	?-?
Objeto	?-+	??-
Fim	--?	+++

Vetor de Estado

Posição	Cat
1	S
2	V
3	O

Dicionário

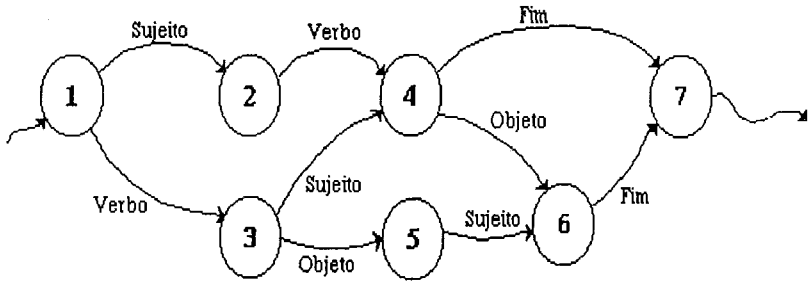
Palavra	Lista_Cat
eu	Sujeito,Obj.
Luiza	Objeto, Suj.
amo	Verbo

b) REPRESENTAÇÃO EM RVG

FIGURA 5.3 - LINGUAGEM SVO

Para se representar a mesma linguagem SVO mas com liberdade de ordem entre Sujeito e Verbo, é necessário quase duplicar o numero de transições do autômato anterior, conforme mostra a figura 5.4a. Enquanto que a alteração necessária para se alcançar o mesmo resultado em RVG é simplesmente passar o vetor *cond* da produção *Verbo* de (-,+,?) para (?,+,?). Com isso, a ocorrencia (-) ou não(+) do Sujeito antes do Verbo passa a não importar(?).

Uma linguagem SVO completa, com liberdade de ordem de Sujeito, Verbo e Objeto, exige 22 transições - quatro vezes o numero original. O equivalente RVG apenas relaxa outra restrição, mudando *cond* de *Objeto* de (?,-,+) para (?,?,+).



a) REPRESENTAÇÃO EM AEF

Produções

Cat	Cond	Result.
Sujeito	+??	-??
Verbo	?+?	?-?
Objeto	?-+	??-
Fim	--?	+++

Vetor de Estado

Posição	Cat
1	S
2	V
3	O

Dicionário

Palavra	Lista_Cat
eu	Sujeito,Obj.
Luiza	Objeto, Suj.
amo	Verbo

b) REPRESENTAÇÃO EM RVG

FIGURA 5.4 - LINGUAGEM SVO LIVRE DE ORDEM SV

A compacidade de RVG se deve à propriedade que este possui de que para cada nova categoria criada só é necessário criar uma produção. Em contra partida, uma categoria pode gerar várias transições em autômatos finitos ou recursivos. RVG se torna mais útil quanto maior for a relação entre numero de transições e categorias, ou seja, gramáticas com poucas categorias em comparação com as transições entre estados são mais compactas em RVG do que em AEF ou AR.

Outra propriedade importante é que em AEF não se pode passar restrições ou informações de estado para estado. A única forma de fazê-lo é criar caminhos separados para cada restrição que se queira propagar. A combinação de todas as restrições pode levar a explosão no tamanho do modelo de autômatos.

Apesar de RVG ser tecnicamente equivalente a AEF, as maneiras de se operar o match e change são diferentes. Em vez de um match de identidade, exato,

faz-se parcial, com os valores '?'. Também não se substitui totalmente o RESC por outro, mas altera-se algumas das dimensões, enquanto outras se propagam inalteradas para os estados seguintes.

5.3.3.3. Produções não Léxicas

Um analisador sintático em RVG, da esquerda para direita, aceita uma palavra sempre que existir uma produção cuja categoria corresponda a da palavra em questão e cujo vetor Condição satisfaça o RESC. Em seguida, é atualizado o RESC utilizando-se o vetor Resultado numa operação de change e a palavra é consumida da entrada.

É possível também, em RVG, a análise de uma palavra sem que esta seja consumida. Para tanto, existe o tipo não léxico de produção. Neste caso, a categoria associada a produção não precisa coincidir com uma das categorias da palavra, mas simplesmente satisfazer o RESC contra o vetor cond e atualizá-lo.

A tabela 5.3 demonstra o uso do tipo léxico(L) e não_léxico(N) para identificar Sintagmas Nominais dentro de sentenças simples do tipo SVO.

Neste exemplo, são formados dois grupos de atributos do vetor de estado: os atributos que identificam os constituintes gramaticais (SVO) e os que identificam os constituintes de um sintagma nominal (artigo, substantivo e nome próprio).

Produções				Vetor de Estado		Dicionário	
Cat	**Lex**	**Cond**	**Result.**	**Posição**	**Cat**	**Palavra**	**Lista_Cat**
Sujeito	N	+???-	-??++	1	S	o	Artigo
Verbo	L	-+??-	?-???	2	V	Luiza	NomePróp.
Objeto	N	?-+?-	??-++	3	O	ama	Verbo
Artigo	L	???++	???-?	4	Artigo	Gui	NomePróp.
Subst	L	????+	???--	5	Cabeça	marido	Subst.
NomePróp.	L	???++	???--				
Fim	L	--??-	+++--				

TABELA 5.3 - SINTAGMA NOMINAL EMBUTIDO

As produções *Sujeito* e *Objeto* habilitam o atributo *Cabeça* que funciona como um 'flag' e que distingue as produções gramaticais das produções do sintagma. As produções gramaticais só voltam a ser satisfeitas quando *Cabeça* é desabilitado e isto só ocorre quando o sintagma nominal é concluido - ocorrência do substantivo ou nome próprio.

As produções de um grupo de categorias não alteram o vetor de estado nos atributos associados a categorias do outro grupo. *Artigo, Subst* e *NomePróprio* não alteram os atributos *Sujeito, Verbo* e *Objeto* e vice-versa ('?' no vetor result e cond).

Este é um exemplo de restrição descontínua, o que é facilmente obtido em RVG. Neste caso, a análise do sintagma nominal é permitida apenas durante a análise do sujeito ou objeto.

Este efeito não é facilmente obtido em AEF. A dificuldade está em interromper a análise do sujeito ou objeto, desviar para a análise do sintagma nominal e voltar para o ponto de desvio. A única solução é replicar o número de transições para o sintagma nominal em cada ponto do autômato onde ele possa acontecer. Outra solução, mas que fugiria ao modelo de AEF, é introduzir-se sub-redes recursivas, mas isso aumentaria a complexidade computacional, o que não ocorre com RVG.

5.3.3.4. Perguntas com o Pronome Interrogativo

Um exemplo mais elaborado de restrição descontínua está contido em sentenças interrogativas com os pronomes 'Que', 'Quando', 'Quem', etc. Neste caso, existe a omissão de um sintagma nominal que é percebida e propagada por toda a análise até ser detetado. Por exemplo :

Quem ama Luiza?	(ausência de Sujeito)
Quem Luiza ama?	(ausência de Objeto)
Quem as pessoas pensam que Luiza ama?	(ausência do Compl.)

Produções				Vetor de Estado		Dicionário	
Cat	**Lex**	**Cond**	**Result.**	**Posição**	**Cat**	**Palavra**	**Lista_Cat**
Sujeito	N	+???--	-???++	1	S	o	Artigo
Verbo	L	-+??--	?-????	2	V	Luiza	NomePróp.
Objeto	N	-?+?--	??-?++	3	O	ama	Verbo
Artigo	L	????++	????-?	4	Gap	Gui	NomePróp.
Subst	L	?????+	????--	5	Artigo	marido	Subst.
NomePróp.	L	????++	????--	6	Cabeça	pensam	Verbo
Conj.Subor	L	?-+?--	+++?--			as	Artigo
PronInter	L	+??---	???+??			pessoas	Subst.
NGap	N	???+++	???---			quem	PronInter.
Fim	L	--?---	+++---			que	Conj.Subor

TABELA 5.4 - PRONOME INTERROGATIVO

Na tabela anterior está a representação RVG para análise destas sentenças. A produção *PronInter* liga o atributo *Gap* que se propaga pela rede até ser desligado pela produção *NGap.* Esta última ocorre em substituição aos constituintes do sintagma nominal que desempenhariam o papel de sujeito, objeto ou complemento nominal. A produção *Fim* só ocorre após a ocorrência de *NGap* pois o atributo *Gap* está desabilitado (-) no vetor condição. Todas as demais produções simplesmente propagam o valor do atributo *Gap*, ou seja, '?' nos os vetores Condição e Resultado.

O reconhecimento das sentenças anteriores ocorre da seguinte maneira:

*-PronInter:*Quem; *Sujeito:NGap:Verbo:*ama; *Objeto:NomePróp:*Luiza; *Fim:*?;
*-PronInter:*Quem; *Sujeito:NomePróprio:*Luiza; *Verbo:*ama; *Objeto:NGap: Fim:*?;
*-PronInter:*Quem; *Sujeito:Artigo:*as; *Subst:*pessoas; *Verbo:*pensam; *ConjSubor:*que; *Sujeito:NomePróp:*Luiza; *Verbo:*ama; *Objeto:NGap: Fim:*?;

5.3.3.5. Frases Embutidas

Embutidura de frases foi a primeira evidência que levou Chomsky a afirmar a necessidade de gramáticas livre de contexto para análise de LN. Mais tarde, ficou claro que o desempenho humano na compreensão destas estruturas é bastante

limitado. As sentenças: *O rato que o gato perseguiu fugiu* ou *O gato perseguiu o rato que fugiu,* são facilmente compreendidas por qualquer ouvinte. Se, no entanto, fôr embutita mais uma frase em ambos os casos, nota-se logo uma diferença quanto a esta facilidade: *O rato que o gato que o cão mordeu perseguiu fugiu* não é mais tão óbvia quanto *O cão mordeu o gato que perseguiu o rato que fugiu.* Por estes exemplos pode-se observar que uma frase pode estar embutida em outra de duas maneiras diferentes: no centro ou na extremidade.

As primeiras sentenças de cada um dos dois exemplos anteriores apresentam frases embutidas de centro com objeto relativo enquanto as duas últimas, embutidas de extremidade com sujeito relativo. Um ouvinte possui uma limitação na sua capacidade de entender sentenças embutidas, principalmente com objeto relativo.

Baseado nesta limitação, Blank propôs que RVG também não fosse ilimitadamente capaz de embutir sentenças no centro, contudo, aproveitasse o poder de representação inerente de uma gramática regular (AEF) para reconhecer embutidas na extremidade, sem restrição de número. Esta imposição decorre da dimensão finita do registro de estados - RESC - normalmente igual a três.

		RESC
Nível	---->	Vetor de atributos Principal
		Vetor de atributos Primeira Embutida
		Vetor de atributos Segunda Embutida

O uso do RESC em três níveis não aumenta a complexidade do modelo inicialmente proposto, podendo ser demonstrado que existe um outro equivalente usando um só nível. O algoritmo continua portanto apresentando tempo O(n).

A seleção sobre qual nível do RESC serão feitas as operações de match e change é tirada do campo 'ação' da produção, que pode indicar um 'Subir' ou 'Descer' no nível do RESC.

A tabela 5.5 expande a gramática apresentada no último exemplo de maneira a aceitar sentenças embutidas. Com as alterações introduzidas, é possível reconhecer o exemplo a seguir, onde se tem uma frase embutida na extremidade de outra que por sua vez está embutida no centro de uma terceira.

As chaves que isolam as barras que interligam as unidades possuem controle.

Produções					Vetor de Estado	
Cat	**Lex**	**Ação**	**Cond**	**Result.**	**Posição**	**Cat**
Sujeito	N	X	+???---??	-???+++??	1	Sujeito
Verbo	L	X	-+??---??	?-???????	2	Verbo
Objeto	N	X	?-+?---??	??-?+++??	3	Objeto
Conj.Subor	L	X	?-+?---??	+++?---??	4	Gap
PronInter.	L	X	+??-----?	???+?????	5	Artigo
NGap	N	X	???++++??	???----??	6	Cabeça
Artigo	L	X	????+++??	????-????	7	FimSN
Subst.	L	X	?????++??	????--???	8	PronRel
NomePróp.	L	X	????+++??	????---??	9	FimEmbc
Embc	N	Desce	?+????+-?	+++----++		
Embd	N	X	?-????+??	+++----+?		
PronRel	L	X	+??----+?	???+?????		
FimSN	N	X	??????+??	??????-??		
FimEmb	N	Sobe	--?-----+	?????--?		
Fim	N	X	--?------	+++------		

Dicionário	
Palavra	**Lista_Cat**
as	Artigo
chaves	Subst.
unidades	Subst.
barras	Subst.
controle	Subst.
interligam	Verbo
possuem	Verbo
isolam	Verbo
que	PronRel

TABELA 5.5 - FRASES EMBUTIDAS

Duas produções, *Embc e Embd,* tratam as sentenças embutidas de centro e de extremidade. Quando estas produções ocorrem, são re-habilitadas as categorias *Sujeito, Verbo* e *Objeto,* permitindo que estas categorias ocorram novamente no contexto da nova frase. Uma vez que a ocorrência de *Embd* caracteriza o término da frase que vinha sendo analisada e o início de uma nova, a atualização (change) do RESC é feita no nível corrente. Isto não acontece no caso de *Embc.* Uma frase embutida no centro começa antes que a frase corrente termine, obrigando *Embc* deslocar o nível de RESC antes de atualizá-lo. O estado é, assim, salvo para ser retomado quando do término da frase embutida que começa. Este estado é retomado pela produção *FimEmb* que restaura o nível anterior do RESC.

Dois atributos do vetor, *FimSN* e *FimEmbc,* são usados como 'flags' para identificar o fim do sintagma nominal e da frase embutida de centro respectivamente.

A frase exemplo seria processada da seguinte maneira:

*-Sujeito:Artigo:*As; *Subst:*chaves; *Embc:PronRel:*que; *Sujeito:NGap: Verbo:*isolam; *Objeto:Artigo:*as; *Subst:*barras; *Embd:PronRel:*que *Sujeito:NGap:Verbo:*interligam; *Objeto:Artigo:*as; *Subst:*unidades; *FimSN:FimEmbc:Verbo:*possuem; *Objeto:Subst:*controle; *Close:.*;

CAPÍTULO VI

GRAMÁTICA LIVRE DE CONTEXTO BASEADA EM RVG

Uma complexidade algoritmica linear no processamento de Linguagem Natural é um resultado que não pode ser desprezado na escolha de um modelo para representar uma gramática natural. RVG oferece esta complexidade linear.

Não obstante, existem outros aspectos a serem levantados.

O primeiro está no fato de alguns trabalhos, entre eles [REED 87], já apresentarem algoritmos com complexidade $O(n^3)$ para análise de gramáticas livres de contexto, onde n é o número de palavras. Estes algoritmos possuem um fator de atenuação f entre 5-10 que, para os valores usuais de n, os aproxima de $O(n^2)$. Este também é um resultado considerável, principalmente se observado o poder de representação adicional conseguido frente ao modelo regular do RVG puro.

Outro aspecto consiste na possibilidade de ambiguidades estruturais da gramática não serem razoavelmente analisadas devido ao modelo finito de representação e mesmo de implementação que aquele propõe. O despreso de uma ou mais destas ambiguidades pode significar a diferença entre reconhecer ou não a sentença analisada.

Existe, ainda, um aspecto prático. Caso se alcance uma dimensão do vetor de estado grande, pode-se cometer tantos erros na implementação ou codificação da gramática em RVG, que a utilização da técnica deixa de ser uma vantagem para se tornar um martírio.

Gramáticas Frase-Estruturadas (GFE), como DCG e ATN, podem parecer uma opção para esses impasses. Tais opções, infelizmente, também apresentam inconvenientes. GFE em geral são tão pobres para tratar restrições descontinuas quanto AEF. O reconhecimento desta dificuldade é comprovado pela proposta de

"Generalized Phrase-Structured Grammar - GPSG" em [GADZAR 85] que além de contorná-la apenas parcialmente, se utiliza de algoritmos $O(n^3|G|^2)$, onde G é o tamanho da gramática em número de produções. Outros algoritmos, até exponenciais, são também utilizados - [BLANK 89].

ATN usa diagramas de transição que representam bem restrições locais, enquanto as descontínuas, envolvem a atribuição, passagem e teste de registros. Se não forem impostas restrições sobre o uso dos registros, mesmo complexidade polinomial torna-se impossível. Junta-se a isso a falta de compacidade de GFE e AEF em geral, e retorna o impasse.

Como então, aproveitarmos o melhor destes dois modelos? Neste capítulo é apresentado um modelo de RVG Recursivo, apoiando-se nos resultados de J.H. Reed de que complexidade $O(n^3)$ é possivel e satisfatória. Esta apresentação segue a descrição de uma metodologia de conversão de GFE, mais especificamente Redes de Transição, para RVG. Por último, são feitas considerações sobre a equivalência das gramáticas em ATN e em RVG Recursivo.

6.1. TRADUÇÃO GFE PARA RVG

Conforme já foi mencionado, a implementação da gramática em RVG pode se tornar uma tarefa complicada devido a dificuldade de se visualisar os constituintes e a sequência destes dentro deste formalismo.

Qualquer pessoa, mesmo sem muita familiaridade com o modelo de AEF, seria capaz de entender o diagrama da figura 5.4a. O mesmo não acontece com a figura 5.4b, que apresenta o mesmo autômato da figura anterior representado em RVG. Neste caso é necessário um mínimo de explicação e adaptação.

Seria melhor, então, se a codificação das regras básicas da gramática obedecessem uma metodologia mais intuitiva e as restrições descontinuas ou locais,

concordâncias, e outras regras da LN, que são facilmente representadas em RVG, pudessem ser introduzidas depois, numa segunda etapa.

O formalismo adotado para essa codificação inicial da gramática é o de produções de uma GFE. A partir deste, é feita uma tradução para RVG para só então, serem inseridas as demais regras e restrições.

6.1.1. Algoritmo de Tradução

Em [REED 89] é apresentado um algoritmo de conversão de GFE para RVG, mas não se chega a um resultado ótimo. Nem o número de produções nem o de atributos do vetor são minimizados. O resultado final obtido está a seguir.

Gramática Frase-Estruturada	**Gramática RVG**		
Aux --> modal	Atributos = modal,not1,have1,not2,be1,not3,have2,be2 e close		
Aux --> modal not	Aux: inic +-+-+----		
Aux --> modal have		CONDIÇÃO	RESULTADO
Aux --> modal not have	modal	+????????	-+----+++
Aux --> modal have be	not1	?+????????	------+++
Aux --> modal not have be	have1	??+??????	---+---++
Aux --> modal be	not2	???+?????	-------++
Aux --> modal not be	be1	????+????	-----+--+
Aux --> have	not3	?????+???	--------+
Aux --> have not	have2	??????+??	-------++
Aux --> have be	be2	???????+?	--------+
Aux --> have not be	close	????????+	
Aux --> be			
Aux --> not be			

TABELA 6.1 - CONVERSÃO SEGUNDO [REED 89]

Será utilizado esse mesmo exemplo de contituinte sintático da língua inglesa, verbo Auxiliar, na apresentação de um algoritmo que minimiza o número de atributos e produções da gramática. Será assumido também, que a passagem da notação de regras de produção para a Rede de Transição (RT) correspondente é um problema resolvido e de fácil solução.

Antes de se apresentar o algoritmo, a nomenclatura adotada doravante será:

Ocorrências de Categoria -arcos da Rede de Transição com mesmo rótulo mas que levam à estados com subredes diferentes.

Categorias de mesmo tipo-arcos da RT com o mesmo rótulo, independente do índice da ocorrência.

Índice da Ocorrência -numerações subescritas ao rótulo do arco e que caracteriza uma ocorrência da categoria. A cada ocorrência é atribuido um índice e categorias com uma única ocorrência dispensam o índice.

Atributo do vetor a_j - *j*-ésimo atributo nos vetores de condição e resultado.

Produção p_i - *i*-ésima produção da gramática.

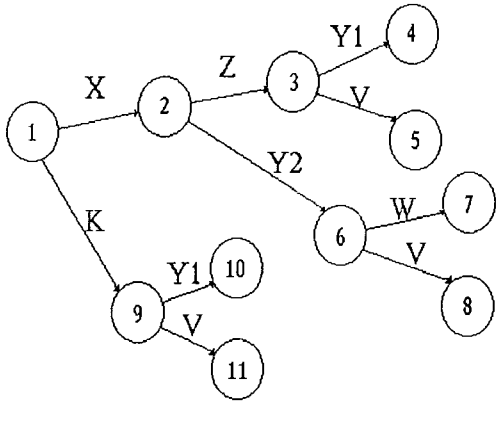
Instância de Ocorrência -uma das aparições da ocorrência da categoria na RT.

Categoria Ascendente -qualquer categoria que apareça antes, seguindo-se a orientação do dígrafo, de qualquer instância de uma ocorrência.

Categoria Descendente -idem para 'aparição depois na RT'.

Categoria Colateral -idem para 'ao lado'.

Por exemplo, na rede de transição abaixo:



Y_1 é uma categoria de Y
Y_1 e Y_2 são categorias do mesmo tipo Y
X, Z e K são pais (ascendentes) de Y_1
Z, $Y_{1,2}$, W e V são filhos (descendentes) de X
W e V são irmãos (colaterais)
Z e Y_2 também
Y_1 filho de Z é uma instância da ocorrência de categoria Y_1
Y_1 filho de K é outra instância da mesma ocorrência

Uma vez estabelecida a nomenclatura, o proximo passo é aplicar um algoritmo de conversão da notação Frase-Estruturada da tabela 6.1 para a rede de transição da figura abaixo, onde se encontram identificadas as categorias e suas ocorrências.

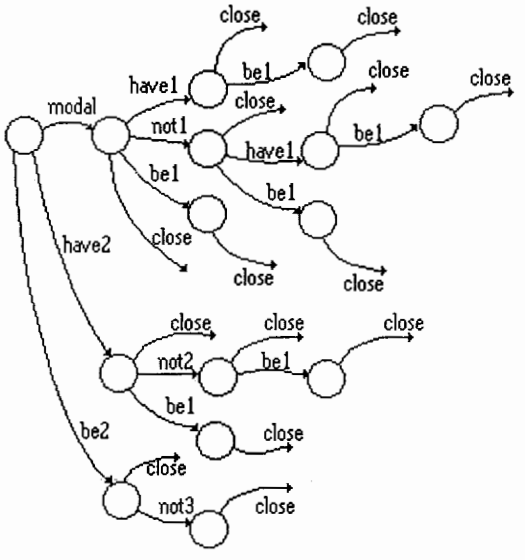


FIGURA 6.1 - REDE DE TRANSIÇÃO VERBO AUXILIAR (INGLÊS)

Monta-se então, uma tabela de produções com os vetores condição e resultado preenchidos da seguinte maneira:

a) o atributo a_j no vetor de condição da produção p_i contém um '-' se a categoria que ele representa é ascendente da categoria que p_i representa sem ser, no entanto, nem irmã nem descendente desta.

b) a_j no vetor condição conterá um '+' se a categoria que ele representa é a mesma de p_i ou então, não é ascendente da categoria de p_i e é descendente ou sòmente colateral desta.

c) a_j de condição conterá um '?' nos casos contrários a estes.

d) a_j no vetor resultado conterá um '+' se a categoria que representa for descendente da categoria da produção p_i.

e) a_j em resultado conterá um '-' se a categoria não for filha de p_i.

	CONDIÇÃO	RESULTADO
	atributos = modal,have1,have2,be1, be2,not1,not2,not3 e close	
modal	++++++???	-+-+-+--+
not1	-+?+?+???	-+-+----+
not2	??-+??+??	---+----+
not3	????-??+?	--------+
have2	+?+++?+??	---+--+-+
be2	+?+?+??+?	-------++
have1	-+?+?????	---+----+
be1	-+-+?????	--------+
close	-?-?-???+	

TABELA 6.2 - VETOR DE ATRIBUTOS APÓS 1ª ETAPA DO ALGORITMO

Feita a distribuição de valores nos vetores de atributos, deve-se passar para a unificação das categorias de mesmo tipo:

f) dois atributos associados a duas ocorrências de categoria podem ser unificados sempre que as ocorrências associadas não forem nem ascendente nem descendente uma da outra.

g) duas produções associadas a duas ocorrências de categorias podem ser unificadas sempre que os atributos associados a essas ocorrências puderem ser unificados e que os filhos que as ocorrências não têm em comum não pertencerem ao mesmo tipo de categoria de nenhum dos pais da outra ocorrência.

Resta ao procedimento estabelecer os critérios para o preenchimento do atributo e produção relacionados com o fim do constituinte *Aux* (*Close*).

h) os atributos *Close* dos vetores condição conterão '?' para todas produções. Nos vetores resultado será preenchido '+' para as produções associadas às categorias terminais e '-' para as demais.

i) finalmente, o vetor condição da produção *Close* é obtido da unificação dos vetores resultado de todas as produções terminais.

As unificações nos vetores condição e resultado obedecem às seguintes equações lógicas:

a_{cond} = '+' Se AND $\forall i$ (AND ($\forall j$ a_{ji})) = '+' ou
categoria de a_j é a mesma que de p_j
= '-' Se AND $\forall i$ (AND ($\forall j$ a_{ji})) = '-' e
OR $\forall i$ (AND ($\forall j$ a_{ji})) = '-'
= '+' caso contrário

a_{resul} = '+' Se AND $\forall i$ (OR ($\forall j$ a_{ji})) = '+'
= '-' Se AND $\forall i$ (OR ($\forall j$ a_{ji})) = '-' e
OR $\forall i$ (OR ($\forall j$ a_{ji})) = '-'
= '+' caso contrário

onde, a_{ji} é o atributo da ocorrência j da produção i,
AND('?', '+') = '?', OR('+', '-') = '+',
AND('?', '-') = '?', OR('+', '+') = '+',
AND('+', '-') = '-', OR('-', '-') = '-',
AND('+', '+') = '+',
AND('-', '-') = '-'.

No exemplo, todas as categorias podem ser unificadas nos atributos e apenas o *Not* pode ser unificado nas produções, obtendo-se:

	CONDIÇÃO	RESULTADO
	modal,have,be, not e close	
modal	++++?	-++++
not	???+?	-??-+
have2	+++??	--+++
be2	+?+??	---++
have1	-+???	--+-+
be1	-?+??	----+
close	-???+	

TABELA 6.3 - GRAMÁTICA FINAL EM RVG

6.1.2. Critérios do Algoritmo

Os passos do algoritmo de conversão GFE $-\triangleright$ RVG estão divididos em três partes:

Critérios de Preenchimento dos Vetores Condição e Resultado

Estes critérios foram obtidos após uma análise do significado dos valores assumidos pelos atributos do vetor de estado, em função do posicionamento das categorias na rede de transição. Este significado está relacionado com a propriedade de RVG de propagar restrições a estados seguintes.

Sempre que uma categoria *a* é alcançada antes de outra *b* e nunca depois ou ao lado desta, o vetor de estado no momento da produção de *b* possui o atributo associado a *a* desabilitado ('-'). Isto significa que existe uma restrição quanto a ocorrência de *a* após e ao lado de *b* e que precisa ser testada no vetor condição de *b*.

Quando a categoria *a* pode ser produzida em concorrência com *b* ou após a produção de *b*, o vetor de estado possui o atributo de *a* habilitado ('+'). Da mesma forma, o vetor de condição da produção *b* deve expressar esta possibilidade testando a sua habilitação.

Além destes dois casos, existe a possibilidade de dúvida quanto a habilitação da categoria. Esta dúvida é provocada pela múltipla instanciação de uma ocorrência de categoria na rede. Assim as relações de ascendência e colateralidade entre *a* e *b* ficam indefinidas, exigindo o relaxamento no teste do atributo associado à categoria ('?').

Para determinar os valores no vetor de resultados a análise é bem mais simples. Parte-se do principio de que uma categoria *b* deve desabilitar a produção de qualquer outra que não seja sua descentente e habilitar as restantes. No caso de habilitação de uma categoria *a*, esta restrição se propaga até que um descendente de *b* não seja ascendente de *a*, só então, a categoria é desabilitada.

Unificação de Ocorrências de Categoria

A segunda parte do algoritmo é composta pelos passos de unificação das diversas ocorrências de categorias do mesmo tipo. Para se entender as condições que determinam a unificação é preciso entender antes, por que existem estas ocorrências da categoria.

Uma ocorrência da categoria define uma sub-árvore da rede de transição cuja raiz é a própria ocorrência. Esta sub-árvore possui ramos que as demais ocorrências de categorias de mesmo tipo não possuem. No exemplo que esta sendo utilizado, a ocorrência *have2* possui um ramo, *not2*, que a diferencia de *have1*, ou seja, existe uma situação na qual o *not* não pode ocorrer após o *have*.

Observando-se a gramática da Lingua Inglesa, constata-se que um verbo modal ('would', 'could', 'must', etc) atrai a negação, impedindo que esta ocorra após o verbo principal ou os verbos auxiliares 'be' e 'have'. Em outras palavras, os contextos diferentes das ocorrências *have1* e *have2* as tornam diferentes.

Numa gramática sensível ao contexto, seria possível representar a categoria *have* de uma maneira única, com uma variável de contexto que diferenciasse as duas ocorrências da categoria. Uma vez que o modelo proposto é livre de contexto, as produções associadas as categorias não podem ser unificadas.

Cada uma das produções *have1* e *have2* testam e geram restrições diferentes devido ao contexto em que ocorrem.

Assim como a categoria *have*, outras duas possuem mais de uma ocorrência: *be* e *not*. A primeira, *be*, se encaixa no mesmo caso de *have* e também não pode ter suas produções unificadas. No entanto, a categoria *not* possui uma característica diferente.

As três ocorrências de *not* diferem nas mesmas categorias que definem os seus contextos: *have* e *be*. Pode-se concluir, portanto, que quem impõe as restrições

não são as ocorrências de *not* em função deste contexto, mas as ocorrências de *have* e *be*. Estas últimas restringem a própria ocorrência após as mesmas.

Por isso, as ocorrências de *not* podem ser unificadas e a única exigência é que na sua produção se ignore as restrições relativas a *have* e *be* e as propague aos proximos estados (vetor Resultado e Condição igual à '?'). Isto já é garantido pelo processo de unificação que é obtido das equações lógicas definidas.

A unificação dos atributos associados às ocorrências de mesmo tipo e a unificação das produções destas ocorrências (toda discussão anterior) só podem ser feitas se nenhuma destas ocorrências fôr produzida no caminho da outra. Nestes casos, as ascendentes definem o contexto das outras e a unificação é impossível.

Por último, as equações lógicas de unificação significam que:

Vetor Condição - Se todos os atributos de todas produções que estão sendo unificados possuem o mesmo valor '+' ou '-', então o atributo unificado (a_{cond}) segue este valor. Se algum dos atributos for diferente dos demais deve-se relaxar ('?') o teste. Isto é, quando da produção de uma categoria, não se tem certeza quanto a restrição à ocorrência da outra categoria associada ao atributo, o teste não é feito.

Vetor Resultado - Se em todas as produções pelo menos um dos atributos fôr igual à '+', o atributo unificado (a_{result}) é '+'. Se em todas as produções nenhum dos atributos fôr igual à '+', o atributo unificado é '-'. Caso contrário, propaga-se a restrição ('?').

Término

A última parte do algoritmo consiste dos passos de preenchimento dos atributos e dos vetores de Condição e Resultado associados à categoria *Close*. Esta categoria é do tipo não_léxico e é alcançada após qualquer categoria cujo estado na

RT seja terminal[8]. Exatamente por este motivo, os atributos associados à *Close* são habilitados em todos os vetores Resultado destas categorias terminais e desabilitados nas demais.

Finalmente, o vetor Condição da produção *Close* é obtido da unificação de todos os vetores Resultado associados a essas mesmas categorias terminais. O motivo, óbvio, é que se as categorias terminais levam ao *Close* a unificação de seus vetores Resultado expressará as concordâncias ('+' ou '-') e discordâncias ('?') nos atributos no momento da produção de *Close*.

6.2. RVG LIVRE DE CONTEXTO

As limitações do modelo original de RVG se devem a intenção de Blank de manter o modelo equivalente a um autômato finito, consequentemente uma gramática regular. Para tornar este modelo representativo de uma gramática livre de contexto, deve-se dotá-lo de recursão. Isto significa, formalmente, a possibilidade de símbolos do conjunto V de variáveis da gramática assumirem qualquer posição no lado direito das produções, ou seja, elementos não terminais podem ser gerados por qualquer produção, antes ou depois de um símbolo terminal.

A transformação de AEF em Autômato Recursivo (Autômato de Pilha) não constitui, por si só, nenhum motivo de preocupação. No entanto, RVG é uma classe de AEF especial, dotada da capacidade de passar restrições de estado para estado. Conforme já foi observado, esta propriedade não é encontrada em outros modelos, mesmo com recursão. É preciso, por tanto, criar um mecanismo que garanta a manutenção desta propriedade.

---

[8] uma categoria α é terminal se $\alpha \in \Sigma$, onde Σ é o conjunto alfabeto(dicionário) da gramática G, α não é terminal se $\alpha \in V$, conjunto de variáveis.

A solução encontrada foi a criação de dois operadores, denominados 'shift' e 'unshift', que mapeiam os atributos de um vetor em outros, num segundo vetor. O vetor de mapeamento contém índices que indicam a posição para onde os atributos do vetor origem devem ser copiados no vetor destino.

Por tanto, dados dois vetores de atributos, f e g, e um vetor de mapeamento m, a operação de shift produz um terceiro vetor de atributos, tal que:

$$\text{shift}(f, g_i, m) \begin{cases} g_i \text{ se } \sim\exists\ m_j \mid m_j = i \\ \text{change}(g_i, f_j) \text{ se } \exists\ m_j \mid m_j = i \end{cases}$$

e $\quad \text{shift}(f, g, m) = \text{shift}(f, g_i, m)$ para todos os i's

Ex_1: shift ((+,-,-,?), (-,-,?), (3,?,?,1)) = (-,-,+)

Porque: m_1 e m_4 contém índices < > ?

$g_{m1} = g_3 \Rightarrow \text{shift}_3 := \text{change}(g_3, f_1)$

$g_{m4} = g_1 \Rightarrow \text{shift}_1 := \text{change}(g_1, f_4)$

Ex_2: shift ((+,-,-,+,+), (+,?,?,-), (?,?,?,2,3)) = (+,+,+,-)

Analogamente, dados dois vetores de atributos, f e g, e um vetor de mapeamento m, a operação de unshift produz um vetor, tal que:

$$\text{unshift}(f_j, g, m_j) \begin{cases} f_j \text{ se } m_j = ? \\ \text{change}(f_j, g_i) \text{ se } m_j <> ? \end{cases}$$

unshift (f, g, m) = unshift (f_j, g, m_j) para todos os j's

Ex_1: unshift ((+,-,-,?,?), (+,+,+,-), (?,?,?,2,3)) = (+,-,-,+,+)

Os dois operadores criados, shift e unshift, são complementares. O propósito do primeiro é passar as restrições, válidas em uma rede, para a sub-rede quando esta é invocada. O segundo, realiza o retorno das restrições para a rede de origem.

Observe a gramática frase-estruturada da tabela 6.4 e sua equivalente RVG Livre de Contexto. Nesta primeira versão, tem-se uma linguagem cujas sentenças são compostas de sintagmas nominal e verbal. O sintagma nominal pode ser um sujeito ou um objeto, enquanto o verbal é um verbo mais um sintagma nominal. Esta gramática é equivalente a utilizada no item 5.3.3.2 e permite a livre ocorrência do Sujeito, Verbo e Objeto.

Na forma em que se encontra, esta gramática não contém, por exemplo, teste de concordância de número entre verbo e sujeito. Para fazê-lo, seria preciso criar um atributo no vetor que seria preenchido pela produção *Sujeito* ou *Verbo*, o que ocorrer primeiro. Este mesmo atributo se propagaria pela rede S até alcançar a produção seguinte, *Verbo* ou *Sujeito.*

GFE
S--> SN SV
SN--> Sujeito
SN--> Objeto
SV--> Verbo S

	TL	CONDIÇÃO	RESULTADO
		SN e SV, init + -	
SN	N	+ -	- +
SV	N	- +	- -
Close	N	- -	- -

	TL	CONDIÇÃO	RESULTADO
		Sujeito e Objeto, init + +	
Sujeito	L	+ ?	- -
Objeto	L	? +	- -
Close	N	- -	- -

	TL	CONDIÇÃO	RESULTADO
		Verbo e SN, init + -	
Verbo	L	+ -	- +
SN	N	- +	- -
Close	N	- -	- -

TABELA 6.4 - RVG-LC SEM CONCORDÂNCIA DE NÚMERO

Este é um exemplo de restrição descontinua que precisa ser propagada, inclusive nas sub-redes. Com os operadores, shift e unshift, pode-se criar o mecanismo para esta propagação.

O RVG Livre de Contexto define uma nova produção como sendo uma sextupla de (cat, cond, result, tipo_lex, vetor_map, vetor_inic) onde os quatro

primeiros campos não diferem da definição anterior. Vetor_map é um vetor de dimensão igual a cond e result, mas preenchido de números, em vez dos valores ternários. Estes números representam os índices no vetor_inic que serão formados a partir do vetor de estado corrente. Vetor_inic é um vetor ternário que contém os valores iniciais dos atributos do vetor de estado para a sub-rede recursiva.

Sentença

	TL	CONDIÇÃO	RESULTADO	MAPEAM.	INIC
		SN, SV e Num, init + - ?			
SN	Sh	+ - ?	- + ?	? ? 3	+ + ?
SV	Sh	- + ?	- - ?	? ? 3	+ - ?
Close	N	- - ?	- - -		

Sintagma Nominal

	TL	CONDIÇÃO	RESULTADO	MAPEAM.	INIC
		Sujeito, Objeto e Num			
Suj. Sg	L	+ ? -	- - -		
Suj. Pl	L	+ ? +	- - +		
Objeto	L	? + ?	- - ?		
Close	Un	- - ?	- - -		

Sintagma Verbal

	TL	CONDIÇÃO	RESULTADO	MAPEAM.	INIC
		Verbo, SN e Num			
Vrb.Sg	L	+ - -	- + -		
Vrb.Pl	L	+ - +	- + +		
SN	Sh	- + ?	- - ?	? ? 3	+ + ?
Close	Un	- - ?	- - -		

TABELA 6.5 - RVG-LC COM CONCORDÂNCIA DE NÚMERO

Deve-se observar que tipo_lex pode assumir, além dos anteriores, um valor que represente a ativação/desativação da rede recursiva (shift/unshift). Estes tipos atribuem à produção um comportamento igual ao tipo 'não_léxico', no que diz respeito ao consumo de palavras da entrada.

A tabela 6.5 apresenta a gramática RVG, da linguagem anterior, já dotada de concordância de número e a figura 6.2, uma execução passo-a-passo da análise de uma sentença exemplo : *Seccionadora apresenta falha.*

Neste exemplo, a dimensão dos vetores em todas as sub-redes é três. Isto deve ser visto com cuidado para não deixar de observar que, por essa coincidência, o índice dos vetores de mapeamento concordam nos valores (3) e sua dimensão e a dos vetores vetor_inic são iguais.

REDE	PASSO	RESC	MAP
S	$RESC_1$:= inic = + - ?	+−?	ϕ
	match SN_S = + - ?	+−?	ϕ
SN	shift ($RESC_1$,map_{SN},$inic_{SN}$)	+−?/++?	??3
	match Suj.Sg = + ? - *(Seccionadora)*	+−?/++?	??3
	change Suj.Sg = - - -	+−?/−−−	??3
	match $close_{SN}$ = - - ?	+−?/−−−	??3
S	unshift ($RESC_1$,map_{SN},$RESC_2$)	+−−	ϕ
	change SN_S = - + ?	−+−	ϕ
	match SV_S = - + ?	−+−	ϕ
SV	shift ($RESC_1$,map_{SV},$inic_{SV}$)	−+−/+−−	??3
	match VrbSg = + - - *(apresenta)*	−+−/+−−	??3
	change VrbSg = - + -	−+−/−+−	??3
	match SN_{SV} = - + ?	−+−/−+−	??3
SN	shift ($RESC_2$,map_{SN},$inic_{SN}$)	−+−/−+−/++−	??3/??3
	match Objeto = ? + ? *(falha)*	−+−/−+−/++−	??3/??3
	change Objeto = - - ?	−+−/−+−/−−−	??3/??3
	match $close_{SN}$ = - - ?	−+−/−+−/−−−	??3/??3
SV	unshift ($RESC_2$,map_{SN},$RESC_3$)	−+−/−+−	??3
	change SN_{SV} = - - ?	−+−/−−−	??3
	match $close_{SV}$ = - - ?	−+−/−−−	??3
S	unshift ($RESC_1$,map_{SV},$RESC_2$)	−+−	ϕ
	change SV_S = - - ?	−−−	ϕ
	match $close_S$ = - - ?	−−−	ϕ

FIGURA 6.2 - EXECUÇÃO DA GRAMÁTICA

6.3. EQUIVALÊNCIA ATNxRVG-LC

ATN é uma técnica bastante usada em PLN. O motivo principal desta preferência está na capacidade deste formalismo de capturar fenômenos gramaticais complexos de uma maneira simples, com o uso de registros. Esta simplicidade, no entanto, oferece um custo em termos de algoritmo, custo esse que está relacionado com as Condições e Ações associadas a cada transição da RT. Essas Condições e Ações do ATN não são formalmente definidas e nem são impostos limites à sua utilização. Apesar disto, na prática, não é utilizado todo o potencial possível de um modelo como este.

No início deste capítulo, foi apresentado um novo formalismo que impõe um custo bem menor e que captura os mesmos fenômenos capturados em ATN. Neste itém, serão apresentados dois destes fenômenos; concordância de número em SN e Voz Passiva, como exemplos de uma metodologia de mapeamento das Condições/Ações do ATN em Vetores Ternários e operações sobre estes, do RVG.

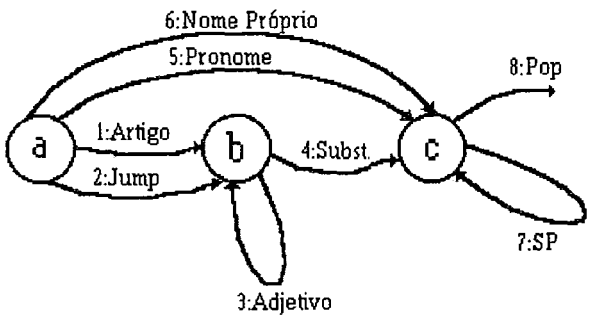
Concordância de Número em SN

Os registros de atributos do ATN têm a função de marcar e propagar as restrições locais ou descontínuas da gramática através do seu preenchimento (unidades de memória). Uma vez que esses registros não assumem, na prática, valores contínuos quaisquer, mas um número finito de possibilidades, pode-se mapeá-los em atributos do vetor ternário. Por exemplo, para se testar, em ATN, a concordância de número dentro de um Sintagma Nominal são empregadas a RT e as Condições/Ações da figura 6.3.

Neste exemplo são usadas apenas duas das possíveis formas (capítulo 5) de Condição/Ação:

A: Preencher atributo com um dos n valores possíveis

C: Testar o atributo contra um dos n valores possíveis



$_{a}$Artigo$_{b}$

A:Preencher Número com Número de *

$_{b}$Substantivo$_{c}$

C:Número está vazio ou é igual à Número de *

A:Preencher Número com Número de *

$_{b}$Adjetivo$_{b}$

C:Número está vazio ou é igual à Número de *

A:Preencher Número com Número de *

$_{a}$Pronome$_{c}$

A:Preencher Número com Número de *

$_{a}$NomPróprio$_{c}$

A:Preencher Número com Número de *

FIGURA 6.3 - SINTAGMA NOMINAL EM ATN

Uma vez que o registro do atributo *Número* só pode assumir dois valores; Plural e Singular, deve-se mapeá-lo em dois atributos e duas produções do RVG. Obtendo-se, então, um atributo e uma produção *Singular* mais um atributo e produção *Plural.* Os atributos tem a função equivalente a armazenar o valor contido no registro do ATN e as produções tem a função de executar a marcação ou teste destes atributos.

Para o mapeamento, é necessário, ainda, que as produções criadas testem o *Número* das palavras sobre as quais as Condições/Ações do ATN atuam, sem consumi-las da entrada. Cria-se para isso um tipo léxico chamado 'Semi-léxico', onde as produções deste tipo precisam "casar" a sua categoria com a da palavra no dicionário, assim como o tipo 'Léxico' o faz, mas não a consome da entrada, assim como tipo 'Não-léxico'.

É preciso, também, que os valores assumidos pelos registros sejam parte da lista de categorias da palavra no dicionário. Por exemplo, a palavra 'seccionadora' estará no dicionário como:

(seccionadora, (substantivo, feminino, singlar))

Por último, deve-se criar um atributo, *Flag,* que serve para obrigar a execução das produções 'Semi-léxicas'. Este flag é marcado por todas as produções que antecedem às categorias associadas com Condição/Ação do ATN e desmarcado pelas produções 'Semi-léxicas'. Com isso, antes de cada categoria que precisa ser testada ou marcada é executada a produção 'Semi-léxica' para fazê-lo.

O equivalente RVG para o SN com concordância de número está na tabela a seguir.

	CONDIÇÃO	RESULTADO
	Art,Adj,Subs,Pron,SP,Sing,Plural e Flag com inic (+ + + + + ?? +)	
Art	+++?+??-	-++-+??+
Adj	?++?+??-	-++-+??+
Subs.	??+?+??-	----+??-
Pron	???++??-	----+??-
SP	----+??-	----+??-
Sing	??????-+	?????+--
Plural	?????-?+	?????-+-

TABELA 6.6 - CONCORDÂNCIA NO SINTAGMA NOMINAL EM RVG

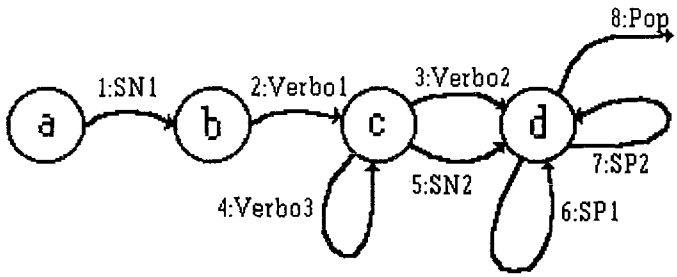
Voz Passiva

O reconhecimento de uma sentença em qualquer voz que o verbo esteja é feito em ATN pela RT da figura 6.4.

Neste caso, existem algumas diferenças em relação ao exemplo anterior. A primeira é que existem Ações relacionadas com a montagem da estrutura profunda da sentença (estrutura interna de representação) além das Ações de interesse gramatical. Estas Ações não serão mapeadas em RVG, pois implicam em se fazer considerações sobre analisadores e não apenas reconhecedores ATN e RVG, o que não é o objetivo deste capítulo, mas que certamente teria equivalência.

A segunda diferença está no fato de que alguns dos testes são feitos em cima de valores marcados em transições anteriores a transição corrente. Por exemplo o

registro *Verbo Principal* é testado em $_{c}Verbo_{d}$ depois deste ter sido marcado em $_{b}Verbo_{c}$.



$_{a}SN1_{b}$
A:Preencher Sujeito com *
$_{b}Verbo_{c}$
A:Preencher Verbo Principal com *
$_{c}Verbo_{c}$
C:Verbo Principal é *ter*
A:Preencher Verbo Aux com Verbo Principal
Preencher Verbo Principal com *
$_{c}Verbo_{d}$
C:* é Participio Passado
Verbo Principal é *ser*
A:Preencher Voz com *Passiva*
Preencher Verbo Aux com Verbo Principal
Preencher Verbo Principal com *
Preencher Objeto Direto com Sujeito
Preencher Sujeito com vazio
$_{c}SN2_{d}$
A:Preencher Objeto Direto com *
$_{d}SP1_{d}$
C:Voz é *Passiva*
Sujeito está vazio
Preposição é *por*
A:Preencher Sujeito com *
$_{d}SP2_{d}$
A:Preencher Modificador com *

FIGURA 6.4 - VOZ PASSIVA/ATIVA EM ATN

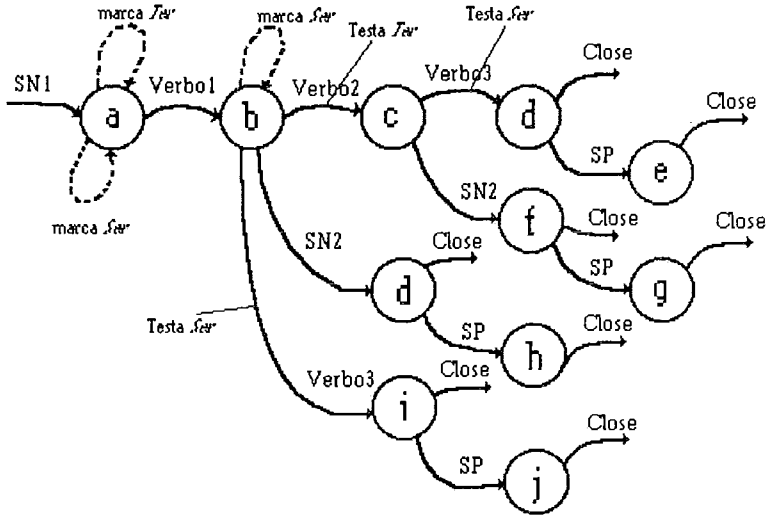
Neste caso, as produções 'semi-léxicas' geradas pelo mapeamento servem, apenas, para marcar o atributo correspondente. As produções normais os testam e desmarcam. Esta diferença em relação ao caso do SN se dá, também, porque estes atributos não se propagam por toda a rede como o *Número* fazia e nem precisam ser testados antes de várias produções.

Outra forma de Condição/Ação não explorada no exemplo anterior é o teste de uma palavra contra algum atributo específico (p.ex. teste do Particípio Passado). Esta forma tem solução trivial, com a colocação deste valor de atributo como categoria da palavra no dicionário e associação da produção com esta categoria.

O único registro do ATN que precisa ser mapeado neste exemplo é o *Verbo Principal.* Este registro funciona como um depositório do último verbo que foi percorrido e possui dois valores possíveis: *Ser* e *Ter.*

Os outros registros usados e que envolvem Condição/Ação só interessam à obtenção da estrutura profunda. Um deles é o registro de *Voz* que combinado com o teste do *Sujeito* e *Preposição* diferencia a ação executada por *SP1* daquela em *SP2.*

O equivalente RVG para o exemplo está na figura a seguir que também apresenta a RT utilizada para sua obtenção com a identificação dos pontos onde são inseridos os testes e marcações dos atributos.



	CONDIÇÃO	RESULTADO
	SN1,SN2,V1,V2,V3,SP,Vser,Vter e Flag com init (+++++++---)	
SN1	++++++??-	-+++++??+
SN2	-+-??+???	-----+???
Verbo1	-+++++??-	-+-+++??+
Verbo2	-+-++++--	-+--++??-
VerboPP	-?-?++-+-	-----+??-
SP	-----+???	------???
Vser	????????+	??????+--
Vter	????????+	??????-+-

FIGURA 6.5 - VOZ PASSIVA/ATIVA EM RVG

Metodologia do Mapeamento

No mapeamento ATN−▷RVG os atributos deste devem ser vistos como variáveis capazes de memorizar o preenchimento ou não de um dos valores assumidos pelo registro do ATN. Graças a pouca variedade destes valores, pode-se criar atributos e produções para cada um destes valores.

As produções do RVG substituem a Ação ou Condição do ATN e com o uso do valor ternário '?' pode-se relaxar estas ações ou condições de maneira a propaga-las pela rede.

De uma maneira geral pode-se dizer que:

a) Para cada valor possível de um registro de ATN, cria-se um atributo do vetor e uma produção semi-léxica em RVG.

b) Para cada registro ATN a ser testado/marcado cria-se um atributo 'Flag'.

c) Os atributos do vetor Condição/Resultado relacionados com o registro ATN são relaxados('?') nas produções normais do RVG e testados/marcados (com '+' ou '-') pelas produções semi-léxicas e léxicas correspondentes as categorias que realizam a Condição/Ação no ATN.

d) As palavras que possuem o atributo do ATN devem ter os valores deste atributo como categoria no dicionário de RVG (isto já acontece em ATN).

e) O atributo de *Flag* é marcado por todas as produções que antecedem o teste do registro e desmarcados pelas produções semi-léxicas ou léxicas que efetivamente o testam.

CAPÍTULO VII

IMPLEMENTAÇÃO

O Sistema Proposto no capítulo IV foi implementado parcialmente. O Sistema de Depuração e Testes já tem toda a parte de aquisição das entradas do processo e a montagem do banco de dados, que será consultado pela Interface Homem-Máquina, implementados e testados [TEIXEI 91]. No Sistema de Desenvolvimento o esforço se concentrou no Processamento de Linguagem Natural, ou seja, o algoritmo de parsing para RVG-LC.

Um registro da gramática da Lingua Portuguesa foi estudado e traduzido para RVG-LC utilizando-se o algoritmo de conversão GFE para RVG. Após esta conversão, o resultado obtido foi aumentado para atender ao formalismo de RVG-LC com a introdução de produções do tipo Shift e Unshift e a identificação de concordâncias de número, gênero e outros fenômenos da Lingua.

As linguagens de programação utilizadas foram o PL/M-86, ASM-86 e Turbo Prolog. As duas primeiras foram utilizadas no SDT devido à facilidade obtida e a necessidade existente de se manusear o hardware envolvido no SDT. Foi considerado também um fator histórico de utilização destas linguagens no projeto do Sequênciador bem como em todo o desenvolvimento do Sistema Digital de Supervisão e Controle dentro do CEPEL.

A escolha da linguagem Turbo Prolog no SD foi decorrência da escolha anterior uma vez que foi desenvolvido, para outros projetos, um Ambiente Operacional Multi-Tarefa e Tempo Real baseado no MS-DOS, [SOUZA 87]. Este ambiente permite a utilização de um programa em Turbo Prolog, como uma das tarefas do Sistema Operacional.

O Turbo Prolog fornece ainda a facilidade de se definir e consultar estruturas de dados em memória de massa. Esta característica era desejada para se

manter as produções e o dicionário de palavras em disco, facilitando assim a alteração e inserção de ocorrências destas.

CAPÍTULO VIII

CONCLUSÕES

O formalismo proposto por Blank, RVG, é uma técnica de Processamento de Linguagem Natural com caracteristicas muito interessantes. A principal virtude desta técnica é sua complexidade mínima diante de um poder de representação bastante abrangente. Existem, conforme foi lembrado, algumas construções gramaticais que RVG deixa de reconhecer e que outras técnicas englobam. Estas construções aparecem principalmente em Linguagens diferentes do Português, p. ex. o próprio Inglês, e mais ainda em Linguas sem origem no Latim.

O custo ao se adotar outra técnica é elevado e a complicação ao implementa-la nem sempre compensa, já que a solução pode ser a adoção de uma Linguagem Natural muito Restrita.

O objetivo do RVG-LC é capturar o melhor do formalismo de Blank e ainda torná-lo mais poderoso em representação. A introdução da recursão aumenta a complexidade algorítmica, mas fornece uma simplicidade de implementação maior além de permitir que construções gramaticais, que antes não eram possíveis, possam ser reconhecidas.

Com a recursão simples perdia-se outra forte característica do modelo original: a habilidade de tratar restrições. Os operadores *shift* e *unshift* mantém esta característica sem introduzir complexidade extra à dos operadores normais *change* e *match.*

Por último, as produções do tipo 'semi-léxico' permitem um pre-análise das palavras em função das categorias ou de uma característica qualquer que se queira testar.

Um resultado colateral obtido da proposta do RVG-LC foi o algoritmo de conversão GFE−▷RVG. Este algoritmo está sendo exercitado com o objetivo de validá-lo totalmente.

Neste trabalho de tese não foi possivel atacar o problema da representação do conhecimento necessária a construção do sistema proposto (SD). No entanto, constou dos trabalhos de implementação, a aquisição deste conhecimento do especialista no processo elétrico, já como preparativo para esta etapa. Esta aquisição foi realizada através de entrevistas que estão gravadas para consulta e que foram resumidas no capítulo 3.

A justificativa para a adoção de LN foi retirada de uma pesquisa bibliografica bastante grande, mas a experiência, por si só, já demonstrava a necessidade de uma forma de programação da lógica de controle mais natural, que não exigisse tanto do especialista e que, principalmente, não introduzisse erro durante a implementação.

BIBLIOGRAFIA

[ALLEN 80] "Analyzing Intention in Utterances"; Allen, J.F.; Perrault, C.R.; Artificial Inttelligence; V15; N3; 1980; pg 143-178.

[ÁRABE 89] "Comunicando-se com Bases de Dados Estatísticos em Linguagem Natural: O Protótipo de uma Interface"; Árabe, A.M.; Anais VI SBIA - PUC/RJ; Nov/89.

[BLANK 85] "A New Kind of Finite - State Automaton: Register Vector Grammar"; Blank, G.D.; In Proceedings of the Ninth Intern. Conf. on AI; Aug/1985.

[BLANK 89] "A Finite and Real-Time Processor for Natural Language";Blank, G. D.; Communications of the ACM; V32; N10; Oct/89.

[BOOTH 67] "Sequential Machine and Automata Theory"; Booth, T.L.; John Wiley and Sons, Inc.; 1967.

[BORLAN 88] "Turbo Prolog 2.0 - Reference Guide & User's Guide"; Borland International; 1988.

[BROWN 75] "Multiple Representations of Knowledge for Tutorial Reasoning"; Brown, J.S.; Burton, R.R.; Representation and Understanding; Bobrow & Collins, Academic Press, New York; 1975; pg 311-349.

[CASANO 86] "Programação em Lógica"; Casanova, M.A.; Giorno, F.A.; Furtado, A.L.; V Escola de Computação; 1986.

[CHANG 73] "Symbolic Logic and Mechanical Theorem Proving"; Chang, C.L.; Lee, R.C.; Academic Press Inc; 1973.

[CHOMSK 57] "Syntactic Structures"; Chomsky, N.; Copyrigth in the Netherlands Mouton & Co., N.V., Publishers, The Hague, 1957; Massachusetts Institute of Technology Press; 10th Printing; 1972.

[CHOMSK 59] "On Certain Formal Properties of Grammars"; Chomsky, N.; Inf. & Control; V2; N2; 1959; pg 137-167.

[CHOMSK 65] "Aspects of the Theory of Syntax"; Chomsky, N.; Massachusetts Institute of Technology Press; 1985.

[CULLIN 78] "Script Application: Computer Understanding of Newspaper Stories"; Cullinford, R.; Ph.D. Th.; Computer Science Dept., Yale Univ.;1978.

[CULLIN 86] "Natural Language Processing - A Knowledge Engineering Approach"; Cullinford, R.; Rownan & Littlefield Publishers; 1986.

[CUNHA 72] "Gramática do Português Contemporâneo"; Cunha, C.; Editôra Bernardo Álvares S.A.; 1972.

[ELETRO 87] "Plano Nacional de Energia Elétrica 1987/2010 - Plano 2010 - Relatório Geral"; Eletrobrás - Centrais Elétricas Brasileiras S.A.; Versão 01 - Preliminar; Mai/87.

[EARLEY 70] "An Efficient Context-Free Algorithm"; Earley, J.; Communications of the ACM; V6; N8; Aug/70; pg 94-102.

[FILLMO 68] "The Case For Case"; Fillmore, C.J.; Universals in Linguistic Theory; Bach & Harms; Ed. Holt, Rinebart and Winston; NY; 1968; pg 1-90;

[FILLMO] "Verbs of Judgings: An Exercise in Semantic Description"; Fillmore, C. J..

[GADZAR 85] "Generalized Phrase-Structured Grammar"; Gadzar, G.; Klein, E.; Pullmum, G.; Sag, I.; Harvard University Press; Cambridge; Mass; 1985.

[GARNER] "Presuposition in Philosophy and Linguistic"; Garner, R..

[HAYES 83] "Tutorial on Techniques and Applications for Natural Language Processing"; Hayes, P.J.; Carbonell, J.G.; Carnegie Mellon Univ.; Oct/83.

[KEENAN] "Two Kind of Presuposition in Natural Language"; Keenan, E.L..

[LANGEN] "The Projection Problem for Presuposition"; Langendoen, T.; Savin, H..

[LEDGAR 80] "The Natural Language of Interactive Systems"; Legard, H.; Whiteside, J.A.; Singer, A. e Seymor, W.; Communications of the ACM; V23, N10; Oct/80.

[LYTINE] "Integrating Syntax and Semantics"; Lytinen, S.; Yale University.

[MANNA 74] "Mathematical Theory of Computation"; Manna, Z.; McGraw-Hill Inc.; 1974

[MINSKY 75] "A Framework for Representing Knowledge"; Minsky, M.; 1975.

[MINSKY 85] "The Society of Mind"; Minsky, M; Touchstone Bks; 1985.

[MIRAND 89] "Sequenciador de Partida, Parada e Operação de Unidades Geradoras de Usinas Hidrelétricas"; Miranda, S.L.; Tese M.Sc.; COPPE/UFRJ; Ago/89.

[NAPIER 89] "Impact of a Restricted Natural Language Interface on Ease of Learning and Productivity"; Napier, H.A.; Lane, D.M.; Batsell, R.R.; Guadango, N.S.; Communications of the ACM; V32, N10; Oct/89.

[OBERME 87] "Natural Language Processing"; Obermeier, K.K.; Byte; Dec/87; pg 225-232.

[PEREIR 80] "Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks"; Pereira, F.C.N.;Warren, D.H.D.; Artificial Intelligence; V13; 1980; pg 231-278.

[REED 87] "An Efficient Context - Free Parsing Algorithm Based on Register Vector Grammars"; Reed, J.; In Proceedings of the Third Annual IEEE Conf. on Expert Systems in Government; 1987.

[REED 89] "Compiling Phrase Structure Grammar Rules Into Register Vector Grammar"; Reed, J.; In Proceedings of the Fifth Annual IEEE Conference on AI Systems in Government; 1989.

[SACCON 87] "Gramática Essencial da Língua Portuguesa"; Sacconi, L.A., Atual Editora Ltda; 1987.

[SCHANK 74] "Inference and the Computer Understanding of Natural Language"; Schank, R.C.; Rieger, C.J.; Artificial Intelligence; V5; 1974; pag 373-412.

[SCHANK 75] "Conceptual Information Processing"; Schank, R.C.; Amsterdam; North-Holland; 1975.

[SCHANK 77] "Scripts, Goals, Plans and Understanding"; Schank, R.C.; Abelson, R.P.; Lawrence Erlbaum Associates Publishers; New Jersey; 1977.

[SCHWAR 74] "Structureless Programming"; Schwartz, J.; Courant Institute, SETL Newsletter; 135A; Jul 1974.

[SEARLE 75] "Indirect Speech Act"; Searle, J.R.; Syntax and Semantics, Volume 3: Speech Act; Cole & Morgan, Eds.; Academic Press; NY; 1975.

[SIMMON 86] "Man-Machine Interfaces: Can they guess what you want?"; Simmons, R.F.; IEEE Expert; 1986.

[SIQUEI 88] "Aspectos Psicolinguísticos Envolvidos na Interação Homem-Máquina em um Sistema de Linguagem Natural'; Siqueira, I.S.P.; Pereira, A.E.; Anais V SBIA - UFRN; Nov/88.

[SOUZA 87] "DOS Multi-Tarefa"; Souza, G. N.; Relatório Técnico; CEPEL Centro de Pesquisas de Energia Elétrica; 1987.

[S&WECo 89] "Expert Systems Service - Stone & Webster Advanced Systems Development Service"; Stone & Webster Engineering Corporation; ISA-90 Conference, New Orleans; 1989.

[TEIXEI 91] "Protótipo do Sistema de Depuração e Testes de Sequênciadores"; Teixeira, L.; Projeto de Final de Curso de Graduação em Engenharia; Escola de Engenharia da UFRJ; 1991.

[TOMITA 85] "Efficient Context-Free Parsing Algorithm for Natural Languages"; Tomita, M.; Proceedings of the Ninith International Conference on AI; UCLA; Aug/85; pg 18-23.

[WEINZE 66] "ELIZA - A Computer Program for the Study of Natural Language Comunication between Man and Machine"; Weinzenbaum, J.; Communication of the ACM; V9; N1; Jan/66; pg 36-45.

[WINOGR 83] "Language as a Cognitive Process"; Winograd, T.; Addison-Wesley Publishing Company; 1983.

[WOODS 70] "Transition Network Grammars for Natural Language Analysis"; Woods, W.; Communications of the ACM; V13, N10; Out/70.

APÊNDICE A

CLASSIFICAÇÃO DAS GRAMÁTICAS SEGUNDO CHOMSKY

Gramática => $G = (V, \Sigma, P)$, onde V Conjunto de Variáveis
Σ Alfabeto ou Dicionário de G
P Conjunto de Produções

Tipo 0 - Irrestrita

$\alpha \text{ -}\triangleright\ \beta$ $\quad \alpha \in (V \cup \Sigma)^+$
$\beta \in (V \cup \Sigma)^*$

Tipo 1 - Sensível ao Contexto

$\alpha \text{ -}\triangleright\ \beta$ $\quad \alpha \in (V \cup \Sigma)^+$
$\beta \in (V \cup \Sigma)^*$
$|\alpha| =< |\beta|$

Ex. S -▷ aSBA
S -▷ abA
AB -▷ BA
bB -▷ bb
bA -▷ ba
aA -▷ aa =▷ $a^n b^n a^n$

Tipo 2 - Livre de Contexto

$\alpha \text{ -}\triangleright\ \beta$ $\quad \alpha \in V$
$\beta \in (V \cup \Sigma)^+$
$|\alpha| = 1$

Ex. S -▷ aSb
S -▷ ab =▷ $a^n b^n$

Tipo 3 - Regular

$A \text{ -}\triangleright\ \sigma B$ ou $A \text{ -}\triangleright\ \sigma$ $\quad A, B \in V$
$\sigma \in \Sigma$

Ex. ba*
(a+b)*
(a+b) + (ab* + b)

APÊNDICE B

SUBCONJUNTO DA GRAMÁTICA PORTUGUÊSA UTILIZADO NO SD

Classes ou Categorias Gramaticais

- Substantivos — Comum; Próprio (Nome Próprio)
- Verbos — Transitivo: Direto; Indireto; Direto e Indireto - Bitransitivo; Intransitivo; Ligação
- Artigos — Definido; Indefinido
- Numerais — Cardinal; Ordinal
- Pronomes — Substantivo/Adjetivo; Pessoal, Possessivo, Demonstrativo, Indefinido, Interrogativo e Relativo
- Advérbios — Tempo; Modo; Lugar; Intensidade
- Adjetivos
- Preposições — Acidentais/Essenciais
- Conjunções — Coordenativas; Subordinativas
- Interjeição

Constituintes do Periodo (Sentença)

S - Sentença	O - Oração
SN - Sintagma Nominal	SV - Sintagma Verbal
SP - Sint. Nom. Preposicionado	GAdj - Grupo/Locução Adjetival
GAdv - Grupo/Locução Adverbial	GS - Grupo Substantival

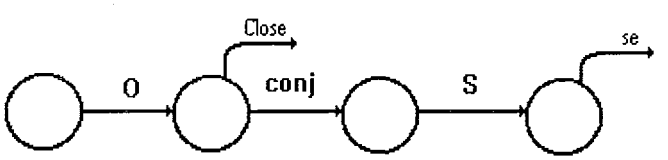
Casos das Categorias Gramaticais

- Substantivo — Gênero (Masculino/ Feminino); Número (Singular/ Plural); Grau (Normal/ Aumentativo/ Diminutivo)

- Artigo	Gênero
	Número
- Adjetivos	Gênero
	Número
	Grau
- Pronome	Gênero
	Número
	Pessoa (Primeira/ Segunda/ Terceira)
- Verbo	Número
	Pessoa
	Tempo (Presente/ Passado/ Futuro)
	Modo (Imperativo/ Subjuntivo/ Indicativo)
	Voz (Ativa/ Passiva/ Reflexiva)
	Formas (Infinitivo/ Gerúndio/ Particípio)
- Advérbio	Grau

Codificação da Gramática Portuguesa em RVG-LC

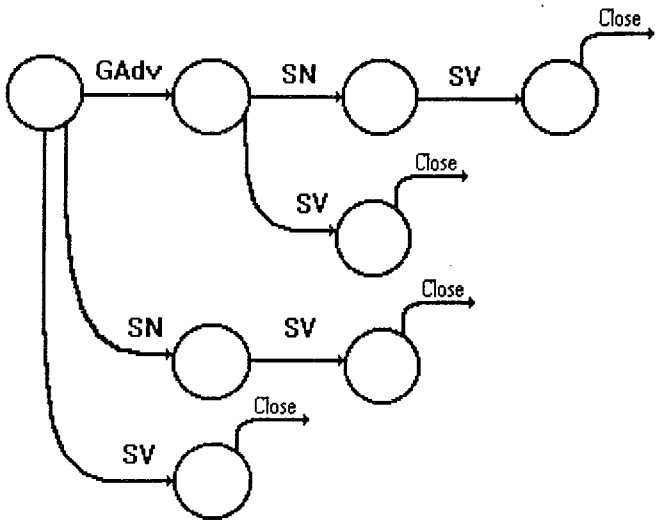
Sentença .



S -▷ O conj S
S -▷ O

	TL	CONDIÇÃO	RESULTADO	MAPEAM.	INIC
		O, Conj, S e close com inic (+- - -)			
O	Sh	+--?	-+-+	????	+++-
Conj	L	-+-?	--+-		
S	Sh	--+?	---+	????	+---
Close	N	-?-+			

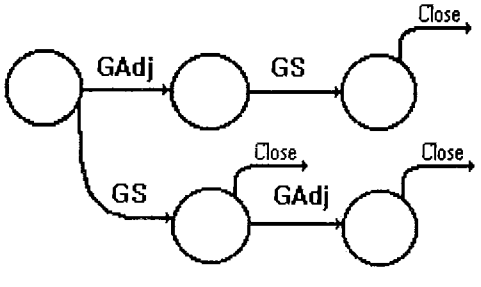
Oração .



O -▷ SN SV
O -▷ GAdv SN SV
O -▷ GAdv SV
O -▷ SV

	TL	CONDIÇÃO	RESULTADO	MAPEAM.	INIC
		GAdv, SN, SV e close com inic (+ + + -)			
GAdv	Sh	+++?	-++-	????	++---
SN	Sh	?++?	--+-	????	++-
SV	Sh	??+?	---+	????	-++++---
Close	Un	---+			

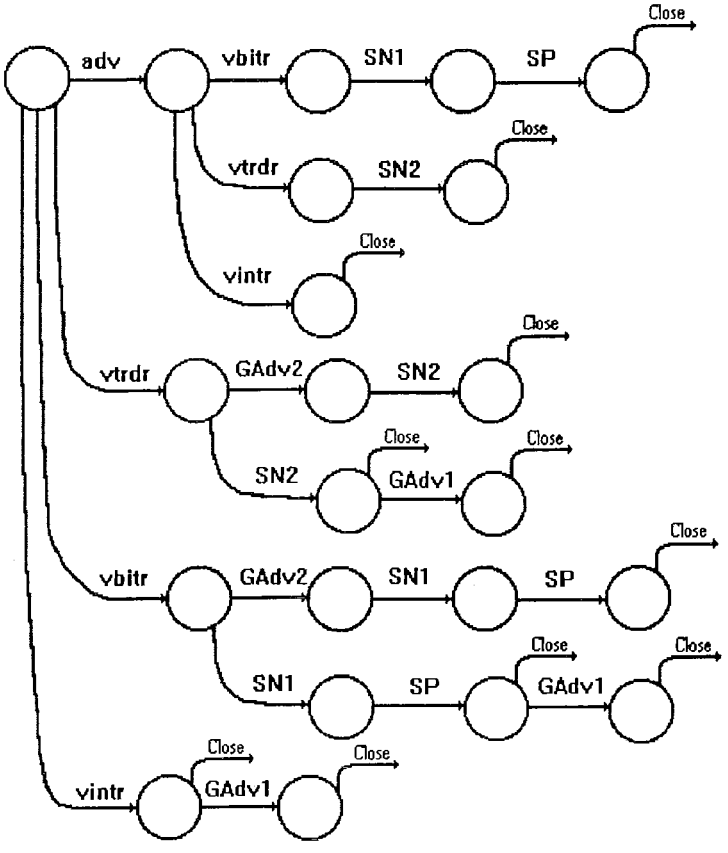
Sintagma Nominal .



SN -▷ GAdj GS
SN -▷ GS GAdj
SN -▷ GS

	TL	CONDIÇÃO	RESULTADO	MAPEAM.	INIC
		GAdj , GS e close com inic (+ + -)			
GAdj	Sh	+??	-??	???	++---
GS	Sh	?+?	?-+	???	+++++----
Close	Un	??+			

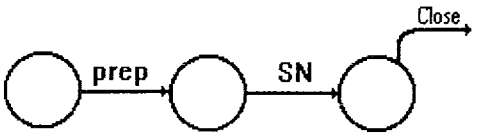
Sintagma Verbal .



SV -▷ adv vintrans
SV -▷ vintrans
SV -▷ vintrans GAdv
SV -▷ vtransdir GAdv SN
SV -▷ adv vtransdir SN
SV -▷ vtransdir SN
SV -▷ vtransdir SN GAdv
SV -▷ vbitrans GAdv SN SP
SV -▷ vbitrans SN SP GAdv
SV -▷ adv vbitrans SN SP
SV -▷ vbitrans SN SP

	TL	CONDIÇÃO	RESULTADO	MAPEAM.	INIC
		GAdv, adv, vbitra, vintra, vtradir, SN, SP e close com inic(-+++++--)			
GAdv1	Sh	+----???	-------+	????????	++---
GAdv2	Sh	+----???	------+--	????????	++---
Adv	L	-++++--?	--+++??-		
Vbitra	L	-?+?????	?----+--		
Vintra	L	-??+????	?------+		
Vtradir	L	-???+???	?----+--		
SN1	Sh	?----+??	?-----+-	????????	++-
SN2	Sh	?----+??	?------+	????????	++-
SP	Sh	?-----+?	?------+	????????	+--
Close	Un	?-----?+			

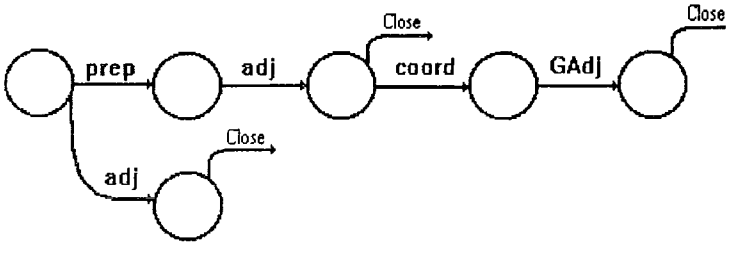
Sint. Nominal com Preposição .



SP -▷ prep SN

	TL	CONDIÇÃO	RESULTADO	MAPEAM.	INIC
		Prep, SN e close com inic (+ - -)			
Prep	L	+−?	−+−		
SN	Sh	−+?	−−+	???	++−
Close	Un	−−+			

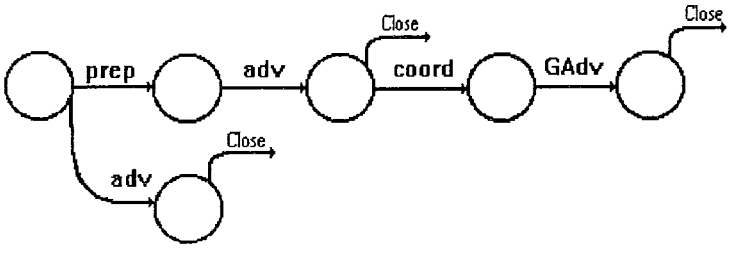
Grupo Adjetival .



GAdj -▷adj
GAdj -▷prep adj
GAdj -▷adj coord GAdj
GAdj -▷prep adj coord GAdj

	TL	CONDIÇÃO	RESULTADO	MAPEAM.	INIC
		Prep, Adj, Coord, GAdj e close com inic (+ + - - -)			
Prep	L	++−−?	−+−−−		
Adj	L	?+−−?	−−+−+		
Coord	L	−−+−?	−−−+−		
GAdj	Sh	−−−+?	−−−−+	?????	++−−−
Close	Un	−−?−+			

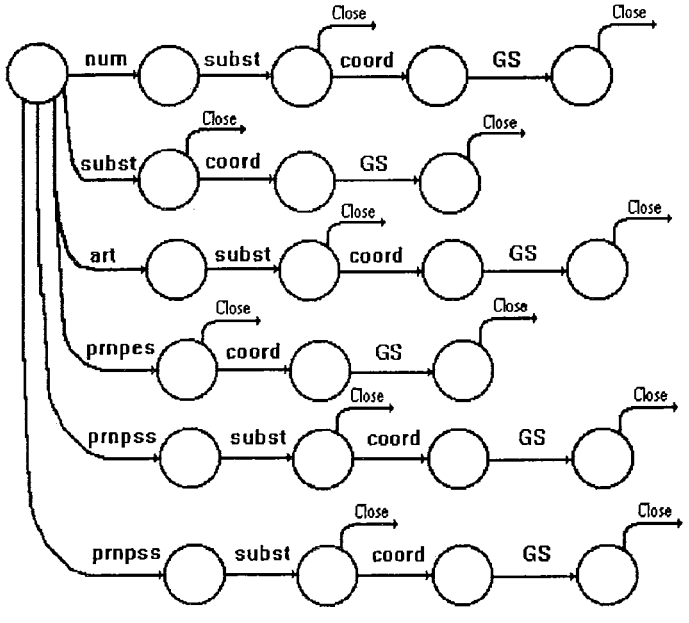
Grupo Adverbial .



GAdv-▷adv
GAdv-▷prep adv
GAdv-▷adv coord GAdv
GAdv-▷prep adv coord GAdv

	TL	CONDIÇÃO	RESULTADO	MAPEAM.	INIC
		Prep, Adv, Coord, GAdv e close com inic (+ + - - -)			
Prep	L	++--?	-+---		
Adv	L	?+--?	--+-+		
Coord	L	--+-?	---+-		
GAdv	Sh	---+?	----+	?????	++---
Close	Un	--?-+			

Grupo Substantival .



GS -▷ num subst
GS -▷ subst
GS -▷ art subst
GS -▷ pronpes
GS -▷ num subst coord GS
GS -▷ subst coord GS
GS -▷ pronpes coord GS
GS -▷ art subst coord GS
GS -▷ pronposs subst
GS -▷ pronposs subst coord GS

	TL	CONDIÇÃO	RESULTADO	MAPEAM.	INIC
		Num, Subst, Art, Pronpes, Pronposs, Prondem, Coord, GS e close com inic (+ + + + + + + - - -)			
Num	L	++++++--?	-+-------		
Subst	L	?+????--?	------+-+		
Artigo	L	++++++--?	-+-------		
Prnpes	L	++++++--?	------+-+		
Prnposs	L	++++++--?	------+-+		
Prndem	L	++++++--?	-+-------		
Coord	L	------+-?	-------+-		
GS	Sh	-------+?	--------+	?????????	++++++---
Close	Un	------?-+			

APÊNDICE C

EXEMPLO DA DESCRIÇÃO DA LÓGICA DE CONTROLE EM LN

O trecho abaixo foi retirado de entrevistas com o especialista no processo elétrico. Estas entrevistas tinham por objetivo fazer-se um levantamento do subconjunto da gramática e vocabulário da Lingua Portuguêsa que eram utilizados em tais descrições.

Durante as entrevistas houve três descrições diferentes relativas à cada manobra de controle. Na primeira descrição o especialista se orientou por um diagrama lógico, na segunda por uma listagem MS e na última pela memória. Ocorreu um fenômeno interessante em relação a cada tipo de orientação. Na orientação por Lógica, o especialista formulava as sentenças do português com uma forte tendencia a sentenças lógicas. Por exemplo, *"Se máquina pronta para partir e freio não aplicado e... então partida da máquina".*

No caso da listagem MS, a tendencia era por máquinas de estados finita; *"O controle vai para o estado pronto para partida se máquina pronta e freio desaplicado e... neste caso abre-se o disjuntor ... ".*

Somente pela lembrança, mas depois de algumas interações, o especialista conseguiu se libertar do vício de linguagem (ou de pensamento).

Estes dois primeiros casos foram considerados inválidos para o propósito do SD pois amarravam a descrição a uma forma que não era natural, mesmo para este especialista, antes de ter contato com Sistemas Digitais de Controle.

As descrições ao final chegaram a seguinte forma:

Partida Automática

"Com o comando de Partida Automática é verificado se a unidade está pronta para partir. A verificação de unidade pronta para partir é feita pela verificação de que algumas entradas estão satisfeitas. Estas entradas são:

Turbina pronta para partir, freio não está aplicado, máquina não está operando como sincrono, não existe bloqueio na máquina, não existe comando de parada da máquina e se existe pressão de ar para freio.

A partir desta verificação das entradas e condições, aunidade estaria pronta para partir e com o comando de Partida Automática seria disparada a partida da máquina.

A partida da máquina seria executada com: comando para abrir válvula esférica, comando para partir bomba de cunha de óleo, comando para abertura do distribuidor e comando para fechamento do distribuidor.

Todas as saidas são atuadas na mesma hora de uma única vez sendo que existe um intertravamento lógico entre elas. O comando de abertura da válvula esférica será o primeiro aser executado e após a abertura da válvula será feita uma permissão para que a Bomba de cunha de óleo parta. Depois que a bomba de cunha de óleo partir seria enviada uma permissão para o distribuidor abrir. A partir da abertura do distribuidor quando a velocidade da máquina for igual à 90% da velocidade nominal seria dado um comando para o regulador de tensão e um comando de fechamento do disjuntor de campo.

Para ser dado o comando de fechamento do disjuntor de campo, antes é feito um teste para saber se a máquina alcançou a velocidade reduzida, só então, é aplicado o comando de fechamento do disjuntor de campo".

APÊNDICE D

LISTAGENS

```
INCLUDE "C:\\IA\\PROLOG\\TESE\\TDOMS.PRO"
INCLUDE "C:\\IA\\PROLOG\\TESE\\TPREDS.PRO"
INCLUDE "C:\\IA\\PROLOG\\TESE\\MENU2.PRO"

DOMAINS
  TERNARY_SYMBOL   = CHAR
  TERNARY_VECTOR   = TERNARY_SYMBOL*
  MAP_SYMBOL       = INTEGER
  MAP_VECTOR       = MAP_SYMBOL*
  WORD             = SYMBOL
  NUM              = SG; PL; NONE
  GEN              = ML; FM; NONE
  TM               = PRES; PAST; FUT; GER; PART; NONE
  LEXICON_FLAG     = LEX; NONLEX

DATABASE - DICIONARIO
  DIC(WORD, SYMBOL, NUM, GEN, TM)

DATABASE - PRODUCOES
  PROD(SYMBOL, LEXICON_FLAG, TERNARY_VECTOR, TERNARY_VECTOR)

PREDICATES
                VALID_SYMBOL(TERNARY_SYMBOL)
                MATCH1(TERNARY_VECTOR, TERNARY_VECTOR)
    NONDETERM   MATCH (TERNARY_VECTOR, TERNARY_VECTOR)
                CHANGE1(TERNARY_VECTOR, TERNARY_VECTOR, TERNARY_VECTOR)
    NONDETERM   CHANGE (TERNARY_VECTOR, TERNARY_VECTOR, TERNARY_VECTOR)
    NONDETERM   SHIFT1(TERNARY_SYMBOL, TERNARY_VECTOR, MAP_SYMBOL,
                        TERNARY_VECTOR)
    NONDETERM   SHIFT (TERNARY_VECTOR, TERNARY_VECTOR, MAP_VECTOR,
                        TERNARY_VECTOR)
    NONDETERM   UNSHIFT1(TERNARY_SYMBOL, TERNARY_VECTOR, MAP_SYMBOL,
                        TERNARY_SYMBOL)
    NONDETERM   UNSHIFT (TERNARY_VECTOR, TERNARY_VECTOR, MAP_VECTOR,
                        TERNARY_VECTOR)
    NONDETERM   STR_TV(STRING, TERNARY_VECTOR)
    NONDETERM   TV_STR(TERNARY_VECTOR, STRING)
    NONDETERM   STR_MV(STRING, MAP_VECTOR)
                ANALYZE(TERNARY_VECTOR, STRING, STRING)
                GOAL_TESTE
```

```
CLAUSES
  VALID_SYMBOL('+') :- !.
  VALID_SYMBOL('-') :- !.
  VALID_SYMBOL('?') :- !.
  VALID_SYMBOL(_)   :- FAIL.
  STR_TV("" , []).
  STR_TV(STR, [H|T])    :- FRONTCHAR(STR, H, STR0),
                           STR_TV(STR0, T).
  TV_STR([] ,"").
  TV_STR([H|T],STR)     :- TV_STR(T,STR0),
                           FRONTCHAR(STR, H, STR0).

  STR_MV("" , []).
  STR_MV(STR, [H|T])    :- FRONTCHAR(STR, S, STR0),
                           CHAR_INT(S, H_ASCII),
                           H=H_ASCII-48,
                           FRONTCHAR(STR0, ',', STR1),!,
                           STR_MV(STR1, T).
  STR_MV(STR, [H|T])    :- FRONTCHAR(STR, S, STR0),
                           CHAR_INT(S, H_ASCII),
                           H=H_ASCII-48,!,
                           STR_MV(STR0, T).

  MATCH1([H1|_], [H2|_]) :- H1=H2.
  MATCH1([H1|_], [H2|_]) :- H1='?'.
  MATCH1([H1|_], [H2|_]) :- H2='?'.
  MATCH1([H1|_], [H2|_]) :- FAIL.

  MATCH([]    , [])    .
  MATCH([H1|_] , [])   :- FAIL.
  MATCH([]    , [H2|_]) :- FAIL.
  MATCH([H1|T1], [H2|T2]):- VALID_SYMBOL(H1),
                           VALID_SYMBOL(H2),
                           MATCH1([H1|T1], [H2|T2]),!,
                           MATCH (T1,T2).

  CHANGE1([H1|_], [H2|_], [H3|_])  :- H2='?', H3=H1.
  CHANGE1([H1|_], [H2|_], [H3|_])  :- H3=H2.

  CHANGE([H1|_] , []    , [_])   :- FAIL.
  CHANGE([]    , [H2|_] , [_])   :- FAIL.
  CHANGE([]    , []    , [])    .
  CHANGE([H1|T1], [H2|T2], [H3|T3]) :- VALID_SYMBOL(H1),
                                VALID_SYMBOL(H2),
                                CHANGE(T1, T2, T3),
                                CHANGE1([H1|T1], [H2|T2], [H3|T3]),!.

  SHIFT1(HF,     [], HM,       _):- FAIL.
  SHIFT1(HF, [HG|TG], HM, [HGA|TGA]):- HM=1,
```

```
                    CHANGE([HG], [HF], [HGA]),
                    TGA=TG,!.
SHIFT1(HF, [HG|TG], HM, [HGA|TGA]):- HMA=HM-1,
                    HGA=HG,
                    SHIFT1(HF, TG, HMA, TGA).

SHIFT (    [], G,  [HM,_], R) :- FAIL.
SHIFT (    [], G,     [], R) :- R=G,!.
SHIFT ([HF|TF], G, [HM|TM], R) :- HM=0,
                    SHIFT(TF, G, TM, R),!.
SHIFT ([HF|TF], G, [HM|TM], R) :- HM<>0,
                    SHIFT1(HF, G, HM, G_ALT),
                    SHIFT(TF, G_ALT, TM, R).

UNSHIFT1(HF,     [], HM, _):- FAIL.
UNSHIFT1(HF, [HG|TG], HM, HR):- HM=1,
                    CHANGE([HF], [HG], [HR]),!.
UNSHIFT1(HF, [HG|TG], HM, HR):- HMA=HM-1,
                    UNSHIFT1(HF, TG, HMA, HR).

UNSHIFT (    [], G,  [HM,_],     R) :- FAIL.
UNSHIFT (    [], G,     [],     R) :- R=[],!.
UNSHIFT ([HF|TF], G, [HM|TM], [HR|TR]) :- HM=0,
                    HR=HF,
                    UNSHIFT(TF, G, TM, TR),!.
UNSHIFT ([HF|TF], G, [HM|TM], [HR|TR]) :- HM<>0,
                    UNSHIFT1(HF, G, HM, HR),
                    UNSHIFT(TF, G, TM, TR).

ANALYZE(SYNSTATE, SENTENCE, SAIDA) :- PROD("CLOSE", _, COND, RESULT),
                    MATCH(COND, SYNSTATE),
                    CHANGE(SYNSTATE, RESULT, NEXTSYNSTATE),
                    WRITE(SAIDA,"CLOSE: ","\N"),!.
ANALYZE(SYNSTATE, SENTENCE, SAIDA) :- PROD("CLOSE", LEX, COND, RESULT),
                    FRONTTOKEN(SENTENCE, W, RESTSENTENCE),
                    DIC(W, "CLOSE", _, _, _),
                    MATCH(COND, SYNSTATE),
                    CHANGE(SYNSTATE, RESULT, NEXTSYNSTATE),
                    WRITE(SAIDA,"CLOSE: ",W,"\N"),!.
ANALYZE(SYNSTATE, SENTENCE, SAIDA) :- PROD(F, NONLEX, COND, RESULT),
                    MATCH(COND, SYNSTATE),
                    CHANGE(SYNSTATE, RESULT, NEXTSYNSTATE),
                    CONCAT(SAIDA, F, SAIDA0),
                    CONCAT(SAIDA0, ": ", SAIDA1),
                    ANALYZE(NEXTSYNSTATE, SENTENCE, SAIDA1).
ANALYZE(SYNSTATE, SENTENCE, SAIDA) :- PROD(F, LEX, COND, RESULT),
                    FRONTTOKEN(SENTENCE, W, RESTSENTENCE),
                    DIC(W, F, _, _, _),
                    MATCH(COND, SYNSTATE),
```

```
                    CHANGE(SYNSTATE, RESULT, NEXTSYNSTATE),
                    CONCAT(SAIDA, F, SAIDA0),
                    CONCAT(SAIDA0, ": ", SAIDA1),
                    CONCAT(SAIDA1, W, SAIDA2),
                    CONCAT(SAIDA2, "\N", SAIDA3),
                    ANALYZE(NEXTSYNSTATE, RESTSENTENCE,SAIDA3).

GOAL_TESTE :- RETRACTALL(_,DICIONARIO),
        RETRACTALL(_,PRODUCOES),
        CONSULT("\\IA\\PROLOG\\TESE\\DIC_RVG.DB", DICIONARIO),
        CONSULT("\\IA\\PROLOG\\TESE\\PRD_RVG.DB", PRODUCOES),
        REPEAT,
        WRITE("\N ENTRE COM A SENTENCA A SER ANALIZADA\N"),
        READLN(STR),
        STR_TV("+++++-----", TV),
        ANALYZE(TV, STR, ""),
        FAIL.

GOAL
  GOAL_TESTE.
```

Nomenclatura

AEF - Autômato de Estados Finito
AP - Autômato de Pilha
AR - Autômato Recursivo
ATN - Augmented Transition Network
CLP - Controladores Lógico-Programáveis
DCG - Definite Clause Grammar
GFE - Gramática Frase-Estruturada
GLC - Gramática Livre de Contexto
LA - Linguagem Artificial
LN - Linguagem Natural
LNR - Linguagem Natural Restrita
MS - Linguagem de Máquinas Sequenciais
RESC - Registro de Estado Sintático Corrente
RT - Rede de Transição
RVG - Register Vector Grammar
RVG-LC - RVG Livre de Contexto
SD - Sistema de Desenvolvimento
SDT - Sistema de Depuração e Testes
SEQ - Sequenciador de Partida, Parada e Operação